全国工程硕士专业学位教育指导委员会推荐教材

软 件 工 程 专 业 核 心 课 程 系 列 教 材

软件测试方法与技术

蔡建平 主编

清华大学出版社

全国工程硕士专业学位教育指导委员会推荐教材
软件工程专业核心课程系列教材

软件测试方法与技术

蔡建平　主编
王安生　修佳鹏　副主编

清华大学出版社
北京

内 容 简 介

本书是作者多年从事软件测试技术研究及课程教学的成果和经验总结。全书共分 5 部分，18 章。第 1 部分(第 1～第 4 章)是软件测试基础，涉及软件测试的一些基本概念和基础知识；第 2 部分(第 5～第 6 章)详细讲述包括静态测试及动态测试在内的软件测试基本方法与技术；第 3 部分(第 7～第 8 章)详细讲述包括缺陷管理、测试过程管理在内的软件测试管理方法与技术；第 4 部分(第 9～第 14 章)详细介绍包括自动化测试、可靠性测试、安全性测试、国际化与本地化测试以及面向对象测试在内的现代软件测试方法与技术；第 5 部分(第 15～第 18 章)是典型应用软件测试，重点介绍 Web、移动、云计算、游戏以及嵌入式等应用的软件测试方法与技术。

本书几乎在每个章节都对支撑该章节软件测试方法和技术应用的测试工具进行了介绍，包括对开源软件测试工具进行了介绍。这些工具将很好地支持高校软件测试课程实践。

本书既可作为软件测试相关课程的研究生(特别是工程硕士专业学位研究生)与高年级本科生的教材，同时还可供软件测试培训和软件测试人员自学参考。

图书在版编目(CIP)数据

软件测试方法与技术/蔡建平主编. —北京：清华大学出版社，2014(2016.2 重印)
软件工程专业核心课程系列教材
ISBN 978-7-302-33925-0

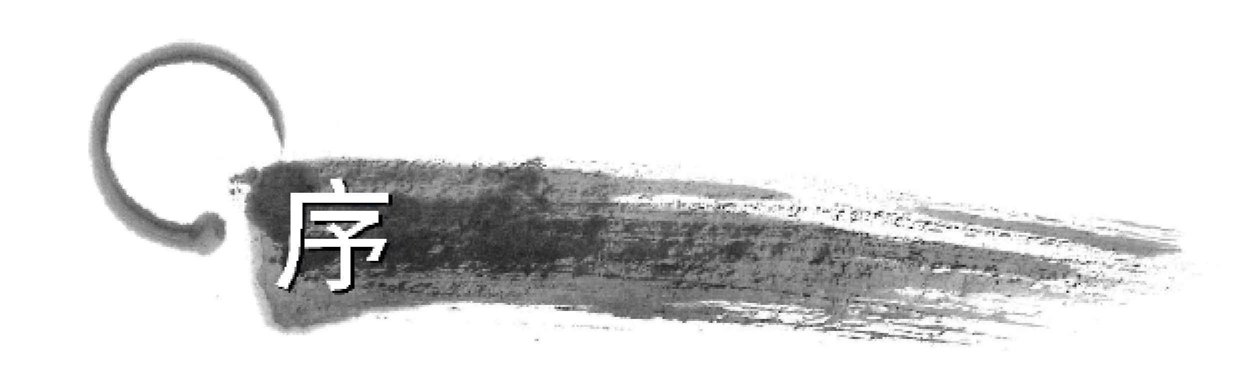
Ⅰ. ①软…　Ⅱ. ①蔡…　Ⅲ. ①软件－测试－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2013)第 220423 号

责任编辑：魏江江　赵晓宁
封面设计：常雪影
责任校对：白　蕾
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：虎彩印艺股份有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：29.25　　**字　　数**：703 千字
版　　次：2014 年 1 月第 1 版　　**印　　次**：2016 年 2 月第4 次印刷
印　　数：2501～2700
定　　价：49.50 元

产品编号：054099-01



序

作为软件工程的重要分支之一，软件测试自20世纪80年代初在我国开展以来，历经20多年的发展，发生了巨大的变化。软件测试已从传统的软件工程瀑布模型中测试阶段的软件测试变化为覆盖包括需求分析、系统设计、详细设计、程序编码、内部测试、系统测试、系统安装、确认验收以及系统维护整个软件工程生命周期的软件测试；从过去单纯的测试概念发展到包括静态分析、质量度量与评价在内的评测结合的软件评测思想；从传统的测试内容分类到基于质量特性、子特性的测试内容分类；从传统的结构化程序测试方法到面向对象的软件测试方法；从早期的单机或桌面测试到网络应用测试及嵌入式应用测试；从以手工测试为主发展到离不开测试工具支持的测试及管理。事实上，软件测试也成为耗费人力、财力和时间的一项复杂的工作，对测试人员提出了高素质、专业化的要求。即软件测试人员不但要精通各种软件测试技术和方法，有一定的软件测试工程实践经验，还要求他们熟悉软件开发技术和软件开发流程，具有快速学习专业知识或领域知识，掌握新技术和应用新工具的能力。另外，软件测试人员要有团队合作意识，善于和人沟通与交流，并能承受被人误解和指责的心理素质。

随着计算机技术的快速发展，软件越来越普遍地应用到各个领域和各个方面，且应用规模越来越大，应用形式越来越复杂，软件质量要求越来越高，软件测试越来越重要。对于高等院校而言，人才的培养是其核心的工作，鉴于高素质的软件测试专业人才越来越奇缺，因此软件测试人才的培养尤为重要和紧迫。国家示范性软件学院的一个重要职责就是要在教学研究、教学实践以及教学改革方面进行大胆的探索和实践，在完善已有课程体系及授课内容的同时，充分利用优秀的教学资源，总结教学经验和科研成果，并编写专业教材为专业课程服务，力争走出一条为国家快速培养高素质软件工程紧缺人才之路。

蔡建平教授长年从事软件工程、软件测试以及软件质量保证的研究、实践和教学，并为编写此书做了较长时间的准备，因此，由他来编写此书应该是具有资格的。该教材具有如下主要特点。

（1）该教材重要知识点的组织和讲述满足国内企业，特别是国内各种评测机构或组织对现代软件测试人才培养的要求。

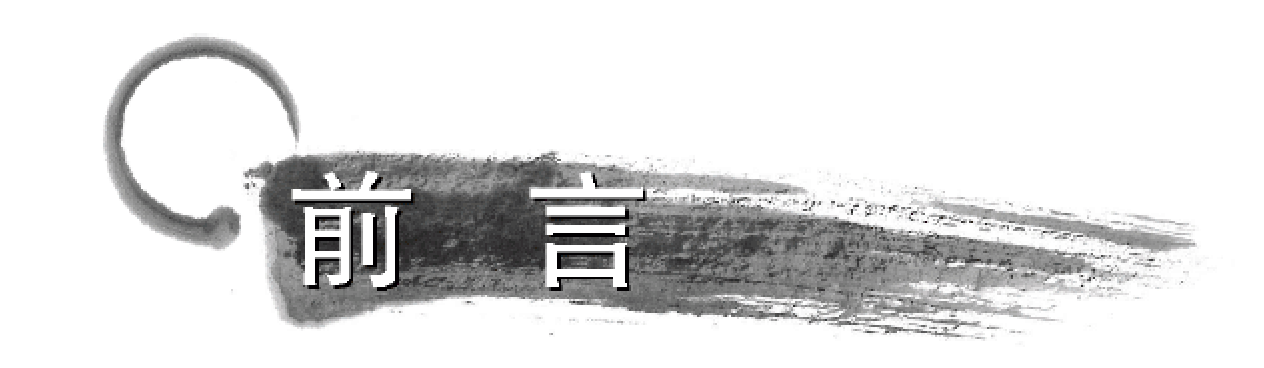
（2）该教材在传统软件测试技术和方法的基础上，强调了软件测试：要与软件质量度量和评价相结合，要满足软件工程全生命周期软件测试的要求，要充分重视软件开发方法和应用方式对软件测试的影响，要注意软件测试工具对软件测试支持的重要作用等。

（3）该书给出了软件测试在几个典型应用领域实际开展的要点和注意事项，这对开阔软件测试人员的眼界、思路和实践有很好的帮助。

总之，该教材在现代软件测试技术的学习、普及、推广和软件测试人才培养以及软件测试教学知识体系的建立等方面进行了有益的探索和尝试。该教材内容全面、翔实，实用性强。该教材的推出，将有益于国内软件测试人员和计算机相关专业的本科生及研究生的学习与能力的培养，有益于推动现代软件测试技术和方法的研究、教学和实践的进一步发展，同时可对我国软件测试业的发展和软件测试紧缺人才的培养起到积极的促进作用。

中国工程院院士　何新贵

2009年6月6日于北京



本书是在《软件测试大学教程》基础上进行了改编。《软件测试大学教程》是2009年9月出版的，到现在为止，已印刷了3次，有20多所高等院校将它指定为教材或学习参考书籍。

《软件测试大学教程》在2012年被北京市教委推荐参评“十二五”普通高等教育本科国家级规划教材，尽管未通过评审，但却于2013年3月被评为全国工程硕士专业学位教育指导委员会推荐教材，也算应了“东方不亮西方亮”的那句成语。

《软件测试大学教程》自发行以来，作者一直密切关注着国内知名的三大电商网站（京东、亚马逊和当当）上读者对该教材的评价。尽管总体评价不错，好评达95%以上，但其中“内容比较空泛、不够详尽”的评价对我触动很大，让我有了改编和再版该教材的想法。

再者，为了满足全国工程硕士专业学位教育指导委员会推荐教材的建设要求，满足适用于研究生、本科生及大专生等不同对象的教学要求，也必须改编和再版该教材。

改编后的教材无论是在深度上、广度上，还是在内容组织和结构调整上做了很大的修改、补充和完善。全书从原来的16章增加到18章（增加了安全测试、移动测试及云测试等技术内容，这些技术内容主要是抛砖引玉，读者可以深入地开展学习和研究），篇幅在原来基础上增加了近50%。

《软件测试大学教程》的改编首先要感谢清华大学出版社的大力支持和帮助，另外，还要感谢北京卓尔教育投资有限公司，最近参与该公司主持的“HP软件测试专业教材开发项目”对教材改编有较大的帮助，很好地做到了基础成果两边有选择地共享。

最后，改编教材的大量内容是取材于互联网，并进行组织和修改的结果。遗憾的是很多网上资料由于转载或引用频繁找不到原出处，在参考文献中无法将原创者准确标注出来，但我在这里要对他们表示感谢。

当然，对家人的感谢是必需的，多年教材、专著的编写离不开她们的支持和照顾。

目前国内软件测试的书籍或教材很多，都有自己的特点或特色。但愿本教材改编后仍保持特色，并更受学生、教师等广大读者欢迎。当然，由于自身能力和水平有限，一定有许多不周到、不准确、遗漏或存在错误之处，恳请读者提出批评和建议，以便再版时修正。

蔡建平

2013年8月31日于北京

主要内容

本书以现代软件测试需求为背景，以现代软件测试方法和技术为基础，以当前软件测试热点应用为典型实例，全面介绍了软件测试的基本概念、软件测试的方法和技术、软件测试管理的方法与技术、高级的或现代的软件测试方法和技术，以及软件测试在当前主流应用中的具体开展和实施。

除了用大量篇幅讲述传统软件测试概念、技术、方法和过程外，书中还详细介绍了全生命周期软件测试的模型概念、软件质量分析、度量和评价（静态测试）方法，现代软件测试的各种技术，以及典型应用（如客户端应用软件、移动应用软件、云应用软件、游戏软件以及嵌入式应用软件等）中的软件测试技术和方法。同时还就软件测试中支持各种测试类型的软件工具选型和使用做了相当全面的介绍，特别是对开源软件测试工具的点评能够开阔读者的学习思路和测试途径。

读者对象

可作为软件测试相关课程的研究生（特别是工程硕士专业学位研究生）与高年级本科生的教材，也可以作为软件测试人员的自学参考书。

本书特色

（1）以现代软件测试思想为指导，除了全面讲述传统软件测试技术和方法外，还通过突出介绍全生命周期软件测试概念、软件质量分析手段、现代软件测试技术、主流测试工具应用以及典型应用测试方法等，帮助学生了解和掌握现代软件测试的各种原理、方法和技术，并能够选择合适的软件测试工具进行相关测试。为培养学生今后成为高素质、专业化的软件测试人才打下基础。

（2）针对软件开发方法和技术的发展变化，针对我国软件外包服务的蓬勃兴起，针对我国国防工业如航空、航天、机械、船舶、电子、通信、石油、化工等大量重要软件或关键软件的实际应用情况和测试要求，特别是对软件高可靠性的要求，选择教材的知识点。

（3）本书对支撑现代软件测试技术应用的测试工具进行了全面的介绍，特别是对开源的软件测试工具的介绍，对高校开设软件测试实验课程是非常有意义的。

（4）本书的内容全面、条理清晰、结构严谨、可用性强，对重点、难点阐述透彻，使其既符合现代软件测试技术发展的潮流，又具有相对的稳定性，还易于剪裁，以满足各类软件测试课程的教学需要和各类软件测试人员的学习需要。

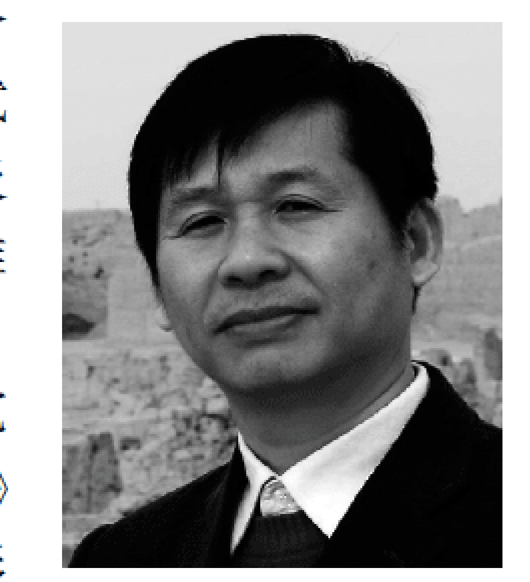
作者简介

蔡建平教授在军队二十多年的工作中，一直从事全军军用共性软件系统项目的论证与研究，取得很多成果。如作为项目负责人获军队科技进步一等奖 1 项（排名第 4）、二等奖 2 项（分别排名第 1 和第 2）、三等奖 2 项（均排名第 1）；负责编著《Ada 程序设计语言高级教程》（解放军出版社，排名第 2）；发表各类学术文章 20 余篇。

在企业工作期间，除负责军工、国防有关单位的软件工程、嵌入式软件测试的技术咨询，并提供解决方案和技术支撑外，还主持开发了 CRESTS（科锐时）系列的嵌入式软件工程和软件测试工具，这些工具已成功地用于航空、航天有关型号项目的测试。

在北京工业大学软件学院任职教授期间，在学科建设、专业建设、实验室建设、“211 工程”

建设、教育部和北京市特色专业建设、科技创新平台建设以及学科交叉(诸如数字艺术、数字体育研究生的人才培养)等方面做了大量的工作,取得了突出成果。作为主要贡献者之一,“面向产业　服务北京　拓展软件工程专业方向的探索与实践”教育教学成果获2009年国家教育教学成果二等奖,排名第3。作为负责人或负责人之一申请数字媒体技术专业,负责多项包括“211”在内的专款建设,主持教育部软件工程(数字媒体技术)特色专业建设以及北京市教委科技创新平台——数字皮影研发平台建设等工作。其中,数字皮影的研究成果成功地用于教学和人才培养,“特色专业建设下数字皮影应用方法的探索”获学校优秀教育教学成果二等奖(排名第1)。“软件测试”及“高级软件编程技术”分别被评为学校精品课程和研究生重点建设课程,其建设成果《软件测试大学教程》、《软件测试实验指导教程》、《软件综合开发案例教程》3部教材已在清华大学出版社出版。其中《软件测试大学教程》被北京市教委推荐参评“十二五”普通高等教育本科国家级规划教材,并于近期被评为全国工程硕士专业学位教育指导委员会推荐教材。



科研上,除了继续在软件工程、嵌入式软件测试等方面开展研究外(主要成果是在清华大学出版社出版了《嵌入式软件测试实用技术》学术专著),还开辟了将学科交叉落到实处的研究领域——“数字皮影表演”。



第1部分　软件测试基础篇

第2部分 软件测试方法与技术基础篇

第 4 部分　高级软件测试方法与技术篇

第1部分

软件测试基础篇

1947 年，计算机还是由机械式继电器和真空管驱动的、有房间那么大的庞然大物，由哈佛大学制造的 MarkII 则是体现当时技术水平的计算机。在一次整机运行中，它突然停止了工作。技术人员爬到计算机上找原因，发现是一只飞蛾受光和热的吸引飞到了计算机内部一组继电器的触点之间，然后被高电压击死。由此，计算机的缺陷产生了，虽然最后该缺陷被技术人员解决了，但是我们从此认识到了它就是缺陷。

如今软件已经渗透到了我们的日常生活中，从办公设备到家用电器，从通信工具到航空航天事业，软件无处不在，然而却又很难完美无缺。

1994 年的秋天，迪斯尼公司发布了第一个面向儿童的多媒体光盘——狮子王动画故事书(The Lion King Animated Storybook)。对此，迪斯尼公司做了大量的宣传。因此，销售额非常可观。然而圣诞节过后，公司接到了大量的投诉电话，称游戏不能运行。经证实，造成这种后果的原因是迪斯尼公司未对市面上使用的许多不同类型的 PC 机型进行测试，软件只能在少数系统中运行。

同样是 1994 年，英特尔奔腾浮点除法缺陷事件，不仅使英特尔公司的形象受到严重影响，并且为自己处理软件缺陷的行为付出了 4 亿多美元的代价。

类似的还有美国航天局火星极地登陆者号探测器事件、爱国者导弹防御系统事件、千年虫问题等，这些事件的后果有的是带来不便，例如游戏玩不成，有的可能是灾难性的——导致机毁人亡。它们的发生都是由于在软件中隐藏着错误。

软件为什么会频繁出问题，如何杜绝或将它们减至最少，影响降至最低呢？在论述软件测试概念之前先介绍一下软件、软件危机及软件工程等概念，然后再讲解软件测试的相关知识。



第1章 软件与软件危机

我们都知道软件的重要意义：软件是信息化的核心，现代国民经济、国防建设、社会发展及人民生活都离不开软件。软件产业是增长最快的朝阳产业，是高投入/高产出、无污染、低能耗的绿色产业。软件产业关系到国家经济和文化安全，体现了国家综合实力，是决定未来国际竞争地位的战略性产业。

软件到底是什么？它具有什么样的特性？它能够干什么？

1.1 软件的概念

随着计算机技术的发展，人们对软件的认识随着阶段的变化而变化。计算机发展的初期，硬件的设计和生产是主要问题，那时的所谓软件，就是程序，甚至是机器指令程序，它们处于从属的地位。软件的生产方式是个体的手工方式，设计是在一个人的头脑中完成的，程序的质量完全取决于个人的编程技巧。其后，人们认识到在机器上增加软件的功能会使计算机系统的能力大大提高，于是在研制计算机系统时既考虑硬件，又考虑软件，而且开始编制一些大型程序系统。这时的生产方式类似于互助合作的手工方式，所以人们认为软件就是程序加说明书。后来，社会需要对计算机提出了更高的要求，某些大型系统的设计和生产的工作量高达几千人/年，指令数百万条，甚至达几千万条，如美国宇航飞船的软件系统有4000万条语句。现在，软件在计算机系统中的比重越来越大，而且这种趋势还在增加。所以人们意识到传统的软件生产方式已不适应发展的需要，于是提出把工程学的基本原理和方法引入到软件设计和生产中，就像机械产品一样，软件生产也被分成几个阶段，每个阶段都有严格的管理和质量检验，科学家们研究出了软件设计和生产的方法，研制出配套的支撑工具，并在设计和生产过程中用书面文件作为共同遵循的依据。这时软件的含义就成了文档加程序。文档是软件的“质”的部分，程序则是文档代码化的形式。

现在软件的正确含义应该是：

(1) 当运行时，能够提供所要求功能和性能的指令或计算机程序集合。

(2) 该程序能够具有满意地处理信息的数据结构。

(3) 该系统能够具有描述程序功能需求以及程序如何操作和使用所要求的文档。

1.1.1 软件特性

软件是人通过智力劳动产生的，软件产品是人的思维结果，是一个逻辑部件，而不是一个物理部件。因此，软件生产水平最终在相当程度上取决于软件人员的教育、训练和经验的积累。所以，软件具有与硬件不同的一些特点。

1. 软件与硬件的不同

软件与硬件的不同或差别主要反映在以下几个方面。

1）表现形式不同

硬件有形，有色，有味，看得见，摸得着，闻得到。而软件无形，无色，无味，看不见，摸不着，闻不到。软件大多存在人们的脑袋里或纸面上，它的正确与否，是好是坏，一直要等到程序在机器上运行才能知道。这就给设计、生产和管理带来许多困难。

2）生产方式不同

软件的开发是智力的高度发挥，而不是传统意义上的硬件制造。尽管软件开发与硬件制造之间有许多共同点，但这两种活动是根本不同的。在两种活动中，通过好的设计能够得到好的质量，但硬件制造阶段可能引入的质量问题在软件开发中却不会出现，反之亦然。这两种活动都依靠人，但人的作用和工作专长之间的关系是完全不同的。因为软件是逻辑产品，如几个人共同完成一个软件项目时，人与人之间就有一个思想交流问题，称之为通信关系。通信是要付出代价的，不只是要花费时间，而且通信中的疏忽常常会使错误增加。人虽然是最聪明的，但人也是最容易犯错误的。

3）要求不同

硬件产品允许有误差，如加工一根轴，其外径精度要求为 Φ500±0.1。生产时，只要达到规定的精度要求就算合格。而软件产品却不允许有误差，要 1 就是 1。如美国金星探测器水手 1 号，导航程序的一个语句的语法正确，但语义错了，结果飞行偏离航线，终于导致试验的失败。又如，阿波罗宇宙飞船飞行控制软件，由于粗心把一个逗号写成了句号，又没能检查出来，几乎造成悲剧性的后果。这就给软件开发和维护，及其质量保证体系提出了很高的要求。

4）维护不同

硬件是会磨损消耗或用旧用坏的，这是因为硬件在使用过程中，由于受到环境的影响（如灰尘、温湿度变化、空气污染、振动等因素）而使产品产生腐蚀或磨损，使硬件故障率增加，甚至损坏，以致不能使用。解决的办法，换上一个相同的备件就是了。而软件不受那些引起硬件损坏的环境因素的影响。因此，在理论上，软件不会用旧、用坏。但实际上，软件整个生存期中，一直处于改变（维护）状态。而随着某些缺陷的改变，很可能会引入一些新的缺陷，导致软件的故障率增加，品质变坏。硬件某一部分变坏，可以使用备用件，而软件则不存在这种备用件，因为软件中任何缺陷都会在机器上导致同样的错误。所以，软件维护要比硬件复杂得多。

2. 软件的特点

软件具有如下特点。

（1）软件是一种逻辑实体，具有抽象性。这个特点使它与其他工程对象有着明显的差异。人们可以把它记录在纸上、内存中或磁盘、光盘上，但却无法看到软件本身的形态，必须通过观察、分析、思考、判断，才能了解它的功能、性能等特性。

（2）软件没有明显的制造过程。软件一旦研制开发成功，就可以大量复制同一内容。也就是说，软件是开发出来的，要对软件的质量进行控制，必须在软件开发方面下工夫。

（3）软件在使用过程中，没有磨损、老化的问题，但有退化问题。软件在生存周期后期不会因为磨损而老化，但会为了适应硬件、环境以及需求的变化而进行修改，而这些修改又不可避免地引入错误，导致软件失效率升高，从而使得软件退化。当修改的成本变得难以接受时，软件就被抛弃。

（4）软件对硬件和系统环境有着不同程度的依赖性。这导致了软件移植的问题。

（5）软件的开发至今尚未完全摆脱手工作坊式的开发方式，生产效率低。

（6）软件是复杂的，而且以后会更加复杂。软件是人类有史以来生产的复杂度最高的工业产品。软件涉及人类社会的各行各业、方方面面，软件开发常常涉及其他领域的专业知识，这对软件工程师提出了很高的要求。

（7）软件的成本相当昂贵。软件开发需要投入大量、高强度的脑力劳动，成本非常高，风险也很大。现在软件的开销已大大超过了硬件的开销。

（8）软件工作牵涉到很多社会因素。许多软件的开发和运行涉及机构、体制和管理方式等问题，还会涉及人们的观念和心理。这些人为的因素，常常成为软件开发的困难所在，直接影响到项目的成败。

1.1.2　软件种类

软件已经渗透到我们的日常生活，被用于各行各业。具体地说，软件可以按如下形式进行分类：

（1）系统软件（如操作系统、数据库管理系统、设备驱动程序、通信处理程序等）。

（2）应用软件（如事务软件、实时软件、工程和科学软件、嵌入式软件、娱乐软件、个人计算机软件、人工智能软件等）。

（3）工具软件（如文本编辑软件、文件格式化软件、磁盘向磁带传输数据的软件、程序库系统以及支持需求分析、设计、实现、测试和支持管理的软件等）。

（4）可重用软件。

1.2　软件危机

前面提到的美国20世纪60年代初飞向金星的第一个空间探测器（水手1号），因其飞船舱内的计算机导航程序之中的一个语句的语义出错，致使偏离航线无法取得成功。这个语句的错误并不是语法错误，而是产生了有悖于程序员所期望的意思。可以称得上世界上最精心设计，并花费了巨额投资的美国阿波罗登月飞行计划的软件，仍然没有避免出错；另外，阿波罗8号太空飞船的一个计算机软件错误，造成存储器的一部分信息丢失；阿波罗14号在飞行的10天中，出现了18个软件错误。当时，软件系统的可靠性得不到保证，几乎没有不存在错误的软件系统。

那时，计算机已有近20年的历史，期间，硬件成本每隔2～3年降低一半，内存和外存的成本每年降低40%左右，硬件性能价格比每十年提高一个数量级，但所需的软件很少能在成本、时间进度、功能规模、维护能力等方面达到要求，特别是可靠性难以符合人们的需要，正如E. E. David指出的那样，大型系统的软件生产已经成为管理人员担惊受怕的项目，于是，人们在20世纪60年代后期惊呼发生了软件危机（software crisis）。

1.2.1　软件危机原因分析

软件危机的原因是多方面的，但不管怎样，软件危机是有它内在的或本质上的原因。下面就软件危机产生的诸多原因进行分析，来揭示软件危机内在的或本质上的原因。

1. 早期编程的特点

从20世纪40年代开始，人们从在MARK-I和ENIAC计算机上编制程序，到软件危机发生时为止的20多年时间里，对软件开发的理解就是编程序，且编程是在一种无序的、崇尚个人技巧的状态中完成的。

同今天的软件开发相比，那时的编程具有一些特点。

1）软件规模相对较小

原因有二：

① 人们对软件可能达到的功能认识有限，那时最为关心的是计算机硬件的发展。作为一个计算机专业人员，他不太关心软件问题（只有为数不多的专业人员才去关心软件），但他必须懂得计算机的结构。作为一个机构，其大量资金也是用于计算机硬件开销上，软件只是作为展现其硬件性能的一种手段而投入少量资金。

② 硬件性能从某种意义上左右着人们对软件的需求，人们总是不自觉地在心中盘算着硬件支持的可能性，这种现象从根本上阻碍了软件的广泛应用，从而限制了软件规模。

2）编程作为一门技艺

大部分软件技术人员不太关心他人的工作，他们往往陶醉于自己的编程技巧，并且那时也无编程规范与标准，这也是由软件规模所决定的。决定软件质量的唯一因素就是该编程人员的素质，时间进度亦是如此。根据 H. Sackman 等人的调查，素质好的人与素质差的人，在软件生产效率上的比例是 10∶1，在程序处理速度和存储容量上的比例是 5∶1。

3）缺少有效方法与软件工具的支持

在当时几乎谈不上有效的编程方法，使用最多的亦只是简单的控制流图。软件工具只有子程序库、装入程序、编辑程序、排错程序以及汇编和编译程序（后来有连接程序）。以致许多程序员沉溺于编程技艺的掌握和使用上。

4）不重视软件开发的管理

由于人们重视个人的技能，再加上软件开发过程能见度低，许多管理人员甚至根本不知道他们的软件技术人员在干什么，究竟做得如何，从而造成管理活动几乎不存在。直到今天，这种观念在一些机构中仍有残留，为此，我们对软件开发的管理问题必须给予更多的重视。

5）软件开发后的维护工作很难进行

由于人们重视个人技能，一旦需要做某些修改，就要原编程人员进行修改。如果他已离开本机构，就需要别人去读懂他的程序，再进行修改和补充。此项工作极其辛苦，说不定修改了一处，反而上百处出现漏洞，变得得不偿失。

上述编程特点导致出现人们常说的软件危机现象。

2. 大型软件开发问题

进入 20 世纪 60 年代，应客观需求需要制作一些大型软件，这样就出现了像第一个空间探测器所描述的例子。国外在开发一些大型软件系统时，遇到了许多困难，有些系统最终彻底失败了；有些系统虽然完成了，但比原定计划推迟了好几年，而且费用大大超出了预算；有些系统未能达到用户当初的期望；有些系统则无法进行修改维护。IBM 公司 OSS/360 系统和美国空军某后勤系统都花费了几千人/年努力，历尽艰辛，但结果令人失望。

随着软件开发应用范围的增广，软件开发规模越来越大。大型软件开发项目需要组织一定的人力共同完成，而多数管理人员缺乏开发大型软件系统的经验，而多数软件开发人员又缺乏管理方面的经验。各类人员的信息交流不及时、不准确，有时还会产生误解。软件项目开发人员不能有效地、独立自主地处理大型软件开发的全部关系和各个分支，因此容易产生疏漏和错误。这也是导致软件危机产生的一个原因。

3. 软件生产的知识密集和人力密集的特点

由于计算机技术和应用发展迅速，知识更新周期加快，软件开发人员经常处在变化之中，软件开发人员不仅需要适应硬件的更新，而且还要涉及日益扩大的应用领域问题研究；软件开发人员所进行的每一项软件开发都必须调整自身的知识结构以适应新的问题的需要，而这

种调整是人所固有的学习行为，难以用工具来代替。

软件生产的这种知识密集和人力密集的特点是造成软件危机的根源所在。从软件开发危机的种种表现和软件开发作为逻辑产品的特殊性可以发现软件开发危机的原因。

4. 用户需求难以明确

对于大型软件往往需要许多人合作开发，甚至要求软件开发人员在软件开发过程中深入应用领域对相关问题进行研究，了解用户需求。这样就需要在用户与软件开发人员之间以及软件开发人员之间相互通信。在此过程中难免发生理解的差异，导致用户需求不明确的问题产生。

用户需求不明确问题主要体现在四个方面：

① 在软件开发出来之前，用户自己也不清楚软件开发的具体需求；

② 用户对软件开发需求的描述不精确，可能有遗漏、有二义性，甚至有错误；

③ 在软件开发过程中，用户还提出修改软件开发功能、界面、支撑环境等方面的要求；

④ 软件开发人员对用户需求的理解与用户本来愿望有差异。这些需求问题将导致后续软件错误的设计或实现，而要消除这些需求上的误解和错误往往需要付出巨大的代价。这同样是产生软件危机的一个原因。

5. 缺乏正确的理论指导，缺乏有力的方法学和工具方面的支持

由于软件开发不同于大多数其他工业产品，其开发过程是复杂的逻辑思维过程，其产品极大程度地依赖于开发人员高度的智力投入。由于过分地依靠程序设计人员在软件开发过程中的技巧和创造性，缺乏正确的理论指导，缺乏有力的方法学和工具方面的支持，从而加剧了软件开发产品的个性化，这也是产生软件危机的一个重要原因。

6. 软件开发复杂度越来越高

软件开发不仅仅是在规模上快速地发展扩大，而且其复杂性也急剧地增加。软件开发产品的特殊性和人类智力的局限性，导致人们无力处理"复杂问题"。所谓"复杂问题"的概念是相对的，一旦人们采用先进的组织形式、开发方法和辅助工具提高了软件开发效率和能力，但随之而来的是新的、更大的、更复杂的问题。

7. 软件危机的本质原因

从前面软件危机的原因分析来看，软件危机产生的本质原因主要有两点：

① 与软件本身的特点有关。

② 与软件的开发人员有关。

从上我们可以看出，软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。这些问题绝不仅仅是不能正常运行的软件才具有的，实际上，几乎所有软件都不同程度地存在这些问题。

1.2.2　软件危机现象

事实上，软件危机包含着两方面的问题：

① 如何开发软件，以满足对软件日益增长的需求。

② 如何维护数量不断膨胀的已有软件。具体地说，软件危机主要有以下一些典型表现。

1. 对软件开发成本和进度的估计常常很不准确

实际成本比估计成本有可能高出一个数量级，实际进度比预期进度拖延几个月甚至几年的现象并不罕见。这种现象降低了软件开发组织的信誉。而为了赶进度和节约成本所采取的一些权宜之计又往往损害了软件产品的质量，从而不可避免地会引起用户的不满。

2. 用户对"已完成的"软件系统不满意的现象经常发生

软件开发人员常常在对用户要求只有模糊的了解，甚至对所要解决的问题还没有确切认

识的情况下，就仓促上阵匆忙着手编写程序。软件开发人员和用户之间的信息交流往往很不充分，“闭门造车”必然导致最终的产品不符合用户的实际需要。

3. 软件产品的质量往往靠不住

软件可靠性度量和评测概念已经建立起来，软件质量保证技术也已成熟，但它们并没有应用到软件开发的全过程中，这些都导致软件产品发生质量问题。

4. 软件常常是不可维护的

很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环境，也不能根据用户的需要在原有程序中增加一些新的功能。“可重用的软件”还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件。

5. 软件通常没有适当的文档资料

计算机软件不仅仅是程序，还应该有完整配套的文档资料。这些文档资料应该是在软件开发过程中产生出来的，而且应该是和程序代码完全一致的最新版本。软件开发组织的管理人员可以使用这些文档资料作为“里程碑”，来管理和评价软件开发工作的进展状况；软件开发人员可以利用它们作为通信工具，在软件开发过程中准确地交流信息；对于软件维护人员而言，这些文档资料更是至关重要、必不可少的。缺乏必要的文档资料或者文档资料不合格，必然给软件开发和维护带来许多严重的困难和问题。

6. 软件成本在计算机系统总成本中所占的比例逐年上升

由于微电子学技术的进步和生产自动化程度不断提高，硬件成本逐年下降，然而软件开发需要大量人力，软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升。美国在1985 年软件成本大约已占计算机系统总成本的 90%。

7. 软件开发生产率提高的速度，既跟不上硬件的发展速度，也远远跟不上计算机应用迅速普及深入的趋势

软件产品“供不应求”的现象使人类不能充分利用现代计算机硬件所具有的能力。

以上列举的仅仅是软件危机的一些明显的表现，与软件开发和维护有关的问题远远不止这些。总之，软件危机一方面是与软件本身的特点有关，另一方面是与软件开发和维护的方法不正确有关。

1.2.3 避免软件危机的方法

如何避免软件危机的产生是一个非常大的课题，是否存在非常有效、十分管用的方法现在还未有结论。以下两点是在软件开发过程中为避免软件危机问题的出现通常要求大家要努力做到的。

(1) 应该对计算机软件有一个正确的认识。应该彻底清除在计算机系统早期发展阶段形成的“软件就是程序”的错误观念，一个软件必须由一个完整的配置组成。事实上，软件是程序、数据及相关文档的完整集合。其中，程序是能够完成预定功能和性能的可执行的指令序列；数据是使程序能够适当地处理信息的数据结构；文档是开发、使用和维护程序所需要的图文资料。1983 年 IEEE 为软件下的定义是：计算机程序、方法、规则、相关的文档资料以及在计算机上运行程序时所必需的数据。虽然表面上看来这个定义列出了软件的五个配置成分，但是，方法和规则通常是在文档中说明并在程序中实现的。

(2) 必须充分认识到软件开发不是某个个体劳动的神秘技巧，而应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。

必须充分吸取和借鉴人类长期以来从事各种工程项目所积累的行之有效的原理、概念、技

术和方法，特别要吸取几十年来人类从事计算机硬件研究和开发的经验教训。极力推广使用在实践中总结出来的开发软件成功的技术和方法，并且研究探索更好、更有效的技术和方法，使用更好的开发工具。这样，在软件开发中就会最大限度地避免软件危机的出现。

1.3 软件工程

1968 年秋季，NATO(北约)的科技委员会召集了近 50 名一流的编程人员、计算机科学家和工业界巨头，讨论和制定摆脱“软件危机”的对策。在那次会议上第一次提出了软件工程这个概念。到 2013 年，软件工程走过了 45 年的历程。

在这 40 多年的发展中，人们针对软件危机的表现和原因，经过不断的实践和总结，越来越深刻地认识到：按照工程化的原则和方法组织软件开发工作，是摆脱软件危机的一个主要出路。

1.3.1 软件工程定义

今天，尽管“软件危机”并未被彻底解决，但软件工程 40 多年的发展仍可以说是硕果累累。下面我们给出一个软件工程的定义后简单讨论一下其所包括的内容。

软件工程是一门研究如何用系统化、规范化、数量化等工程原则和方法去进行软件的开发和维护的学科。它作为一个新兴的工程学科，主要研究软件生产的客观规律性，建立与系统化软件生产有关的概念、原则、方法、技术和工具，指导和支持软件系统的生产活动，以期达到降低软件生产成本、改进软件产品质量、提高软件生产率水平的目标。软件工程学从硬件工程学和其他人类工程学中吸收了许多成功经验，明确提出了软件生命周期的模型，发展了许多软件开发与维护阶段适用的技术和方法，并应用于软件工程实践，取得良好的效果。

1. 软件工程的具体含义

软件工程的具体含义体现在 4 个方面：

(1) 把软件开发看成一个有计划、分阶段、严格按照标准或规范进行的活动(软件工程是指导计算机软件开发和维护的工程学科，软件工程方法＝工程方法＋管理技术＋技术方法)；

(2) 将系统的、规范的、可度量的方法应用于软件的开发、运行和维护的过程(将工程化应用于整个软件的研发过程中，并研究上述提到的途径)；

(3) 要求采用适当的软件开发方法和支持环境及编程语言来表示和支持软件开发各阶段的各种活动，并使开发过程条令化、规范化，使软件产品标准化，开发人员专业化；

(4) 用工程学的观点进行费用估算和计划制定，用管理科学中的方法和原理进行软件生产的管理，用数学的方法建立软件开发中的各种模型和各种算法。

2. 软件工程三要素

软件工程包括三个要素，即方法、工具和过程。

1) 软件工程方法

软件工程方法为软件开发提供了“如何做”的技术。它包括了多方面的任务，如项目计划与估算，软件系统需求分析，数据结构、系统总体结构的设计，具体算法的设计、编码、测试以及维护等。

2) 软件工具

软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。目前，已经推出了许多软件工具，这些软件工具集成起来，建立起称之为计算机辅助软件工程(CASE)的软件开发支撑系统。CASE 将各种软件工具、开发机器和一个存放开发过程信息的工程数据库组合起

来形成一个软件工程环境。

3）软件工程过程

软件工程过程则是将软件工程的方法和工具综合起来以达到合理地、及时地进行计算机软件开发的目的。过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和协调变化所需要的管理及软件开发各个阶段完成的里程碑。

软件工程是一种层次化的技术。任何工程方法(包括软件工程)必须以有组织的质量保证为基础。全面的质量管理和类似的理念刺激了不断的过程改进,正是这种改进导致了更加成熟的软件工程方法的不断出现。支持软件工程的根基就在于对质量的关注。

3. 软件工程基本原理

著名软件工程专家 B. Boehm 综合有关专家和学者的意见并总结了多年来开发软件的经验,于 1983 年在一篇论文中提出了软件工程的七条基本原理。

1）用分阶段的生命周期计划进行严格的管理

统计表明,不成功的软件项目中有 50%左右是由于计划不周造成的。应该把软件生命周期划分为若干阶段,并制定出相应的切实可行的计划,严格按照计划对开发和维护进行管理。B. Boehm 认为,应制定和严格执行 6 类计划：项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划、运行维护计划。

2）坚持进行阶段评审

设计的错误占软件错误的 63%,编码错误只占 37%。而且在后期纠正错误的代价非常高。因此,必须严格坚持阶段评审,及早发现和纠正错误。

3）实行严格的产品控制

在现实中,由于外部原因要求对需求等进行修改是难免的。但必须有严格的管理制度和措施。

4）采用现代程序设计技术

如结构化程序分析、结构化程度设计和面向对象程序设计等。

5）软件工程结果应能清楚地审查

由于软件是一种看不见摸不着的逻辑产品,对它的检验和审查很困难。因此,应提供可视化的检验标准和方法。

6）开发小组的人员应该少而精

软件开发小组的人员应该素质高,人员不宜过多。人员素质低和人员过多,都会导致软件的错误率高,且开发效率下降,成本增加。

7）承认不断改进软件工程实践的必要性

软件工程是一门不断迅速发展的学科,必须学习和跟踪先进的技术和方法,也要不断总结经验、改进方法,要不断进行技术创新。

B. Boehm 指出,遵循前六条基本原理,能够实现软件的工程化生产；按照第七条原理,不仅要积极主动地采纳新的软件技术,而且要注意不断总结经验。

4. 软件工程框架

软件工程的框架可概括为目标、过程和原则。

1）软件工程的目标

软件工程的目标是生产具有正确性、可用性、开销适宜、进度保证,并且项目成功的软件产品。正确性指软件产品达到预期功能的程度；可用性指软件基本结构、实现及文档为用户可用的程度；开销适宜是指软件开发、运行的整个开销满足用户要求的程度；软件项目成功指

开发成本低、功能与性能满足需求、易于移植、维护方便以及按时完成开发任务并及时交付软件产品。这些目标的实现不论在理论上还是在实践中均存在很多待解决的问题，它们形成了对过程、过程模型及工程方法选取的约束。

2）软件工程的过程

软件工程的过程是指生产一个最终能满足需求且达到工程目标的软件产品所需要的步骤，主要包括开发过程、运行过程、维护过程。它们覆盖了需求、设计、实现、确认以及维护等活动。需求活动包括问题分析和需求分析。问题分析获取需求定义，又称软件需求规约；需求分析生成功能规约。设计活动一般包括概要设计和详细设计。概要设计建立整个软件系统结构，包括子系统、模块以及相关层次的说明、每一模块的接口定义；详细设计产生程序员可用的模块说明，包括每一模块中数据结构说明及加工描述。实现活动把设计结果转换为可执行的程序代码。确认活动贯穿于整个开发过程，实现完成后的确认，保证最终产品满足用户的要求。维护活动包括使用过程中的扩充、修改与完善。伴随以上过程，还有管理过程、支持过程、培训过程等。

3）软件工程的原则

软件工程的原则是指围绕工程设计、工程支持以及工程管理在软件开发过程中必须遵循的原则，具体为：

① 采取适宜的开发模型，用以控制易变的需求。

② 采用合适的设计方法，支持软件的模块化、抽象与信息隐藏、局部化、一致性以及适应性等设计要求。

③ 提供高质量的工程支持，特别要强调软件工具和环境对软件过程的支持。

④ 重视开发过程的管理，要有效利用可用的资源、生产满足目标的软件产品、提高软件组织的生产能力等。

5. 软件工程的本质特征

基于软件特性及软件危机产生的原因，我们可以清楚地了解软件工程的本质特征，即：

① 软件工程关注于大型程序的构造。

② 软件工程的中心课题是控制复杂性。

③ 软件经常变化（控制和管理）。

④ 开发软件的效率非常重要（工具与环境）。

⑤ 和谐地合作是开发软件的关键（团队精神）。

⑥ 软件必须有效地支持它的用户。

⑦ 软件工程领域是由一种文化背景的人为另一种文化背景的人创造产品。

6. 软件开发技术和软件项目管理

软件工程包括软件开发技术和软件项目管理两方面内容。

1）软件开发技术与开发方法

软件开发技术是为了完成软件生命周期各阶段的任务，所必须具备的技术手段。软件开发技术包括：①软件开发方法（是一种使用早已定义好的技术集及符号表示习惯来组织软件生产过程的方法，其方法一般表述成一系列的步骤，每一步都与相应的技术和符号相关，目的是在规定的投资和时间内，开发出符合用户需求的高质量的软件）。②软件工具（是为了支持软件人员开发和维护而使用的软件，它可以放大人类的智力、提高工作效率、便于管理实施，并为软件开发方法提供自动的或半自动的软件支撑环境，辅助软件开发任务的完成。当前，在软件开发过程中人们越来越重视工具的使用，用以辅助进行软件项目管理与技术生产）。③软件

开发环境(是将软件生命周期各阶段使用的软件工具有机地集合成为一个整体,形成能够持续支持软件开发与维护全过程的集成化软件支撑环境,用以开发软件,提高开发效率和软件质量,降低开发成本,以期从管理和技术两方面解决软件危机问题)。

在软件开发过程中常用的软件开发方法有以下几种。

(1) 面向数据流的结构化程序开发方法(最终关注程序结构)。

- 指导思想:自顶向下,逐步求精。
- 基本原则:功能的分解与抽象。
- 适合于数据处理领域的问题。

(2) 面向数据结构的开发方法——Jackson 方法。

- JSP(Jackson Structured Programming):首先描述问题的输入输出数据结构,分析其对应性,然后推出相应的程序结构,从而给出问题的软件过程描述。以数据结构为驱动。
- JSD(Jackson Structured Design):首先建立现实世界的模型,再确定系统的功能需求。以事件为驱动,基于进程的开发方法。

(3) 支持程序开发的形式化方法(基于模型的方法)——维也纳方法。

将软件系统当做模型来给予描述,把软件的输入、输出看做模型对象,把这些对象在计算机内的状态看做该模型在对象上的操作。

(4) 面向对象开发方法。

- 基本出发点是尽可能按照人类认识世界的方法和思维方式来分析和解决问题。
- 面向对象方法包括面向对象分析、面向对象设计、面向对象实现。

2) 软件项目管理。

软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理、项目计划等。

3) 软件工程中的技术复审和管理复审

每个阶段结束前要进行技术复审和管理复审。

(1) 技术复审是从技术角度确保软件质量,降低软件成本(尽早发现问题)。审查过程包括准备(如成立审查小组)、简要介绍情况、阅读评审文档、开审查会、返工、复查等。

(2) 管理复审主要是从管理的角度对成本、进度、经费等进行复审,以保证项目正常地开展。

在软件工程理论的指导下,发达国家已经建立起较为完备的软件工业化生产体系,形成了强大的软件生产能力。软件标准化与可重用性得到了工业界的高度重视,在避免重用劳动、缓解软件危机等方面起到了重要作用。

1.3.2 软件生命周期

软件工程的传统解决途径强调使用生命周期方法学和各种结构分析及结构设计技术。它们是在 20 世纪 70 年代为了应付应用软件日益增长的复杂程度、漫长的开发周期以及用户对软件产品经常不满意的状况而发展起来的。

人类解决复杂问题时普遍采用的一个策略就是"各个击破",也就是对问题进行分解然后再分别解决各个子问题的策略。软件工程实际上是从时间角度对软件开发和维护的复杂问题进行分解,把软件生存的漫长周期或把用户的要求转变成软件产品的过程。

1. 什么是软件生命周期

同任何事物一样,一个软件产品或软件系统也要经历孕育、诞生、成长、成熟、衰亡等阶段,

一般称为软件生命周期(软件生存周期)。把整个软件生命周期划分为若干阶段,使得每个阶段有明确的任务,使规模大、结构复杂和管理复杂的软件开发变得容易控制和管理。通常,软件生命周期包括可行性分析与项目计划、需求分析、设计(概要设计和详细设计)、编码、测试、维护等活动,可以将这些活动以适当的方式分配到不同的阶段去完成。

这种按时间划分的思想方法是软件工程中的一种思想原则,即按部就班、逐步推进,每个阶段都要有定义、工作、审查、形成文档供交流或备查,以提高软件的质量。但随着新的面向对象的设计方法和技术的成熟,软件生命周期方法的指导意义正在逐步减少。

2. 软件生命周期的阶段划分

1) 软件生命周期的阶段概念

可以从以下几个方面了解软件生命周期的阶段概念。

(1) 采用生命周期方法学开发软件的时候,从对任务的抽象逻辑分析开始,一个阶段一个阶段地进行开发。

(2) 前一个阶段任务的完成是开始进行后一个阶段工作的前提和基础,而后一阶段任务的完成通常是使前一阶段提出的解法更进一步具体化,加进了更多的物理细节。

(3) 每一个阶段的开始和结束都有严格标准,对于任何两个相邻的阶段而言,前一阶段的结束标准就是后一阶段的开始标准。

(4) 在每一个阶段结束之前都必须进行正式严格的技术审查和管理复审,从技术和管理两方面对这个阶段的开发成果进行检查,通过之后这个阶段才算结束(如果检查通不过,则必须进行必要的返工,并且返工后还要再经过审查)。

(5) 审查的一条主要标准就是每个阶段都应该交出"最新式的"(即和所开发的软件完全一致的)高质量的文档资料,从而保证在软件开发工程结束时有一个完整准确的软件配置交付使用。

(6) 文档是通信的工具,它们清楚准确地说明了到这个时候为止,关于该项工程已经知道了什么,同时确立了下一步工作的基础。此外,文档也起备忘录的作用,如果文档不完整,那么一定是某些工作忘记做了,在进入生命周期的下一阶段之前,必须补足这些遗漏的细节。在完成生命周期每个阶段的任务时,应该采用适合该阶段任务特点的系统化的技术方法,如结构分析或结构设计技术。

2) 软件生命周期阶段划分的意义

软件生命周期阶段划分具有以下意义。

(1) 把软件生命周期划分成若干个阶段,每个阶段的任务相对独立,而且比较简单,便于不同人员分工协作,从而降低了整个软件开发工程的困难程度。

(2) 在软件生命周期的每个阶段都采用科学的管理技术和良好的技术方法,而且在每个阶段结束之前都从技术和管理两个角度进行严格的审查,合格之后才开始下一阶段的工作,这就使软件开发工程的全过程以一种有条不紊的方式进行,保证了软件的质量,特别是提高了软件的可维护性。

总之,采用软件工程方法论可以大大提高软件开发的成功率,软件开发的生产率也能明显提高。

3) 软件生命周期阶段划分的方法

目前划分软件生命周期阶段的方法有许多种,软件规模、种类、开发方式、开发环境以及开发时使用的方法论都影响软件生命周期阶段的划分。在划分软件生命周期的阶段时应该遵循的一条基本原则就是使各阶段的任务彼此间尽可能相对独立,同一阶段各项任务的性质尽可

能相同，从而降低每个阶段任务的复杂程度，简化不同阶段之间的联系，有利于软件开发工程的组织管理。一般说来，软件生命周期由软件定义、软件开发和软件维护三个时期组成，每个时期又进一步划分成若干个阶段。

(1) 软件定义时期的任务是：

① 确定软件开发工程必须完成的总目标；

② 确定工程的可行性，导出实现工程目标应该采用的策略及系统必须完成的功能；

③ 估计完成该项工程需要的资源和成本，并且制定工程进度表。这个时期的工作通常又称为系统分析，由系统分析员负责完成。软件定义时期通常进一步划分成三个阶段，即问题定义、可行性研究和需求分析。

(2) 开发时期的主要任务是具体设计和实现在前一个时期定义的软件，它通常由4个阶段组成：总体设计，详细设计，编码和单元测试，综合测试。

(3) 维护时期的主要任务是使软件持久地满足用户的需要。具体地说：

① 当软件在使用过程中发现错误时应该加以改正；

② 当环境改变时应该修改软件以适应新的环境；

③ 当用户有新要求时应该及时改进软件满足用户的新需要。通常对维护时期不再进一步划分阶段，但是每一次维护活动本质上都是一次压缩和简化了的定义和开发过程。

软件生命周期中软件项目管理是自始至终的，软件项目管理包括软件度量、项目估算、进度控制、人员组织、配置管理、项目计划等。

统计数据表明，大多数软件开发项目的失败，并不是由于软件开发技术方面的原因。它们的失败是由于不适当的管理造成的。遗憾的是，尽管人们对软件项目管理重要性的认识有所提高，但在软件管理方面的进步远比在设计方法学和实现方法学上的进步要小得多，至今还提不出一套管理软件开发的通用指导原则。

3. 软件生命周期模型

任何软件都是从最模糊的概念开始的。从概念提出的那一刻开始，软件产品就进入了软件生命周期。在经历需求、分析、设计、实现、部署后，软件将被使用并进入维护阶段，直到最后由于缺少维护费用而逐渐消亡。这样的一个过程，称为"生命周期模型"(Life Cycle Model)。

典型的几种生命周期模型包括瀑布模型、迭代式模型、快速原型模型等。

1) 瀑布模型

瀑布模型由于酷似瀑布闻名。在该模型中，首先确定需求，并接受客户和软件质量保证(SQA)小组的验证。然后拟定规格说明，同样通过验证后，进入计划阶段……可以看出，瀑布模型中至关重要的一点是只有当一个阶段的文档已经编制好并获得软件质量保证小组的认可才可以进入下一个阶段。这样，瀑布模型通过强制性的要求提供规格说明文档来确保每个阶段都能很好地完成任务。但是实际上往往难以办到，因为整个模型几乎都是以文档驱动的，这对于非专业的用户来说是难以阅读和理解的。但对于应用结构化软件开发方法中大型软件系统的开发，瀑布模型有其天生的优势。

2) 迭代式模型

迭代式模型是RUP(Rational Unified Process，统一软件开发过程，统一软件过程)推荐的周期模型。在RUP中，迭代被定义为：迭代包括产生产品发布(稳定、可执行的产品版本)的全部开发活动和要使用该发布必需的所有其他外围元素。所以，在某种程度上，开发迭代是一次完整地经过所有工作流程的过程。至少包括需求工作流程、分析设计工作流程、实施工作流程和测试工作流程。实质上，它类似小型的瀑布式项目。RUP认为，所有的阶段(需求及其

他)都可以细分为迭代。每一次的迭代都会产生一个可以发布的产品,这个产品是最终产品的一个子集。

3) 快速原型模型

快速原型(Rapid Prototype)模型在功能上等价于产品的一个简单原型。瀑布模型的缺点就在于不够直观,快速原型法就解决了这个问题。一般来说,根据客户的需要在很短的时间内解决用户最迫切需要,完成一个可以演示的产品。这个产品只是实现部分的功能(最重要的)。它最重要的目的是确定用户的真正需求。

软件生命周期模型的发展实际上是体现了软件工程理论的发展。在早期,软件的生命周期处于无序、混乱的情况。一些人为了能够控制软件的开发过程,就把软件开发严格地区分为多个不同的阶段,并在阶段间加上严格的控制和审查,确保质量。否则,在软件生命周期中发现问题和解决问题而付出的代价将会随着时间和阶段的变化成倍或成数量级增加的,如图 1-1 所示。

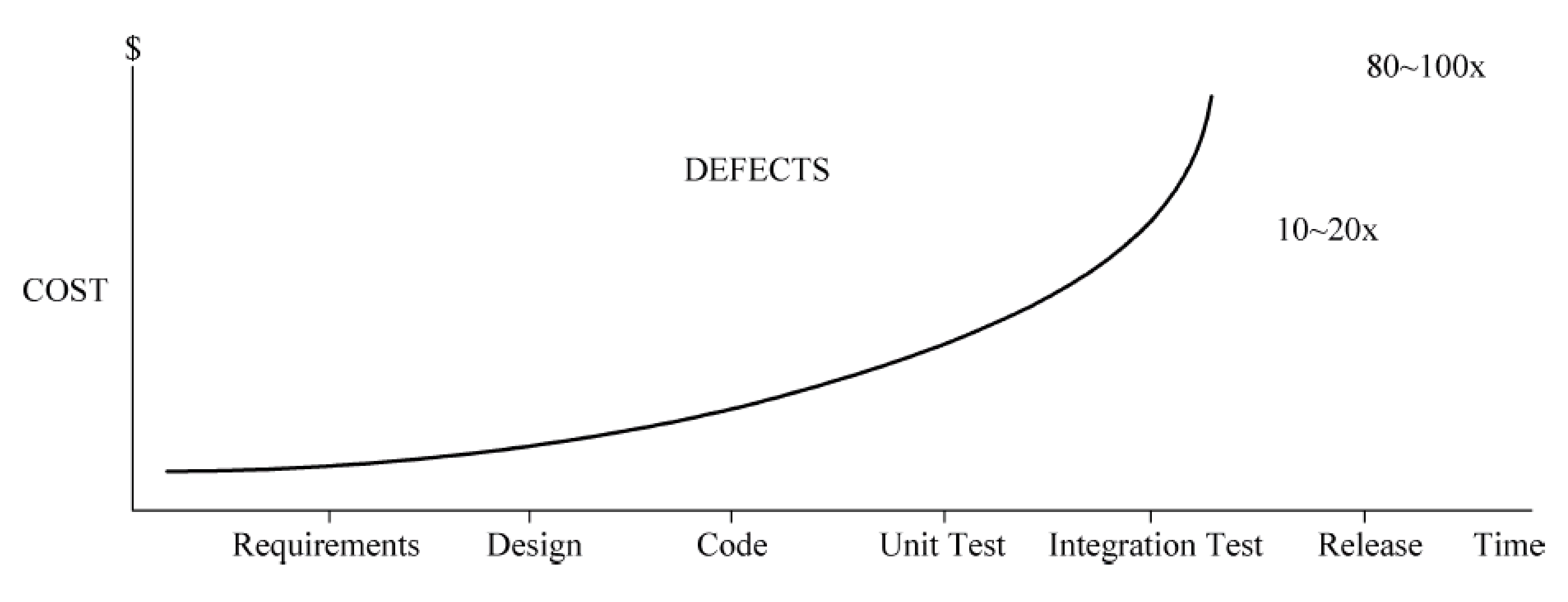


图 1-1 引入同一变化付出的代价随时间和阶段变化的趋势

1.3.3 敏捷开发过程

自从 20 世纪 70 年代,软件工程产生以来,人们提出了很多软件开发方法,这些方法大都强调软件开发过程中必须遵守某些严格的规定。而且随着技术的发展,提出在对需求和技术不断变化的情况下实现快速的软件开发的要求。在 20 世纪 90 年代后期,一些软件开发人员开始强调灵活性在软件开发过程中所发挥的作用,提出一系列新的软件开发方法,这其中包括敏捷方法或敏捷开发。

1. 敏捷开发

敏捷开发被认为是软件工程的一个重要的发展。它强调软件开发应当是能够对未来可能出现的变化和不确定性作出全面反应的。

敏捷开发的总体目标是通过“尽可能早地、持续地对有价值软件的交付”,使客户满意。很多客户都有一些随着时间变化的业务需求,不仅表现在新发现的需求上,也表现在对市场变化做出反应的需求上。通过在软件开发过程中加入灵活性,敏捷开发可以使用户能够在开发周期的后期增加或改变需求。

敏捷开发主要是用于需求模糊或快速变化的前提下,小型开发团队的软件开发活动。敏捷开发能够在保证软件开发成功的前提下,尽量减少开发过程中的活动和产品,做到“刚刚好”,从而在满足所需的软件质量要求的前提下,力求提高开发的效率。

敏捷开发强调:

① 注重个人及互动胜于过程和工具;

② 注重可用的软件胜于详尽的文档;

③ 注重客户协作胜于合同谈判;

④ 注重响应变化胜于恪守计划。

敏捷开发是一种以人为核心、迭代、循序渐进的开发方法。在敏捷开发中,软件项目的构建被切分成多个子项目,各个子项目的成果都经过测试,具备集成和可运行的特征。换言之,就是把一个大项目分为多个相互联系,但也可独立运行的小项目,并分别完成,在此过程中软件一直处于可使用状态。

敏捷开发是针对传统的瀑布开发模式的弊端而产生的一种新的开发模式,是一种面临迅速变化的需求快速开发软件的能力,目标是提高开发效率和响应能力。为达到该目标,敏捷开发定义了 12 条原则:

(1) 最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。

(2) 即使在开发的后期,也不拒绝需求变更(敏捷过程利用变更为客户创造竞争优势)。

(3) 经常交付可工作软件(交付的间隔可以从几个星期到几个月,交付的时间间隔越短越好)。

(4) 在整个项目开发期间,业务人员和开发人员最好是天天在一起工作。

(5) 强化激励机制,为受激励的个人构建项目(为他们提供所需的环境和支持,并且信任他们能够完成工作)。

(6) 在团队内部,最富有效果和效率的信息传递方法是面对面交谈。

(7) 可工作软件是进度的首要度量标准。

(8) 敏捷过程提倡可持续的开发速度(责任人、开发者和用户应该保持一种长期、稳定的开发速度)。

(9) 不断地关注优秀的技能和好的设计,增强敏捷能力。

(10) 尽量简化工作。

(11) 好的架构、需求和设计出自组织团队自身。

(12) 每隔一定时间,团队要反省如何更有效地工作,并相应地调整自己的行为。

敏捷方法有很多具体的方法,常用的敏捷方法有 7 种。

1) XP

XP(极限编程)的思想源自 Kent Beck 和 Ward Cunningham 在软件项目中的合作经历。XP 注重的核心是沟通、简明、反馈和勇气。因为知道计划永远赶不上变化,XP 无须开发人员在软件开始初期做出很多的文档。XP 提倡测试先行,以将后面出现 Bug 的几率降至最低。

2) SCRUM

SCRUM 是一种迭代的增量化过程,用于产品开发或工作管理。它是一种可以集合各种开发实践的经验化过程框架。SCRUM 中发布产品的重要性高于一切。

该方法由 Ken Schwaber 和 Jeff Sutherland 提出,旨在寻求充分发挥面向对象和构件技术的开发方法,是对迭代式面向对象方法的改进。

3) Crystal Methods

Crystal Methods(水晶方法族)由 Alistair Cockburn 在 20 世纪 90 年代末提出的。之所以是个系列,是因为他相信不同类型的项目需要不同的方法。虽然水晶系列没有 XP 那样的产出效率,但还是有很多人接受并遵循它。

4) FDD

FDD(Feature-Driven Development,特性驱动开发)由 Peter Coad、Jeff de Luca、Eric

Lefebvre 共同开发，是一套针对中小型软件开发项目的开发模式。此外，FDD 是一个模型驱动的快速迭代开发过程，它强调的是简化、实用、易于被开发团队接受，适用于需求经常变动的项目。

5）ASD

ASD(Adaptive Software Development，自适应软件开发)由 Jim Highsmith 在 1999 年正式提出。ASD 强调开发方法的适应性(adaptive)，这一思想来源于复杂系统的混沌理论。ASD 不像其他方法那样有很多具体的实践做法，它更侧重为 ASD 的重要性提供最根本的基础，并从更高的组织和管理层次来阐述开发方法为什么要具备适应性。

6）DSDM

DSDM(动态系统开发方法)是众多敏捷开发方法中的一种，它倡导以业务为核心，快速而有效地进行系统开发。实践证明 DSDM 是成功的敏捷开发方法之一。在英国，由于其在各种规模的软件组织中的成功，它已成为应用最为广泛的快速应用开发方法。

DSDM 不但遵循了敏捷方法的原理，而且非常适合那些成熟的有传统开发方法基础的软件组织。

7）轻量型 RUP

RUP 其实是个过程的框架，它可以包容许多不同类型的过程，Craig Larman 极力主张以敏捷型方式来使用 RUP。他的观点是：目前如此众多的努力以推进敏捷型方法，只不过是在接受能被视为 RUP 的主流 OO 开发方法而已。

2．敏捷开发过程

敏捷可用于任何软件过程，实现要点是将软件过程设计为如下方式：允许项目团队调整并合理安排任务，理解敏捷开发方法的易变性并制定计划，精简并维持最基本的工作产品，强调增量交付策略，快速向客户提供适应产品类型和运行环境的可运行软件。因此，敏捷过程很容易适应变化并迅速做出自我调整，在保证质量的前提下，做到文档、度量适度。

下面以极限编程(eXtreme Programming，XP)为例，介绍敏捷开发过程。

XP 使用面向对象方法作为推荐的开发范型。XP 包含了策划、设计、编码和测试 4 个框架活动的规则和实践。图 1-2 描述了 XP 过程，并指出与各框架活动相关的关键概念和任务。

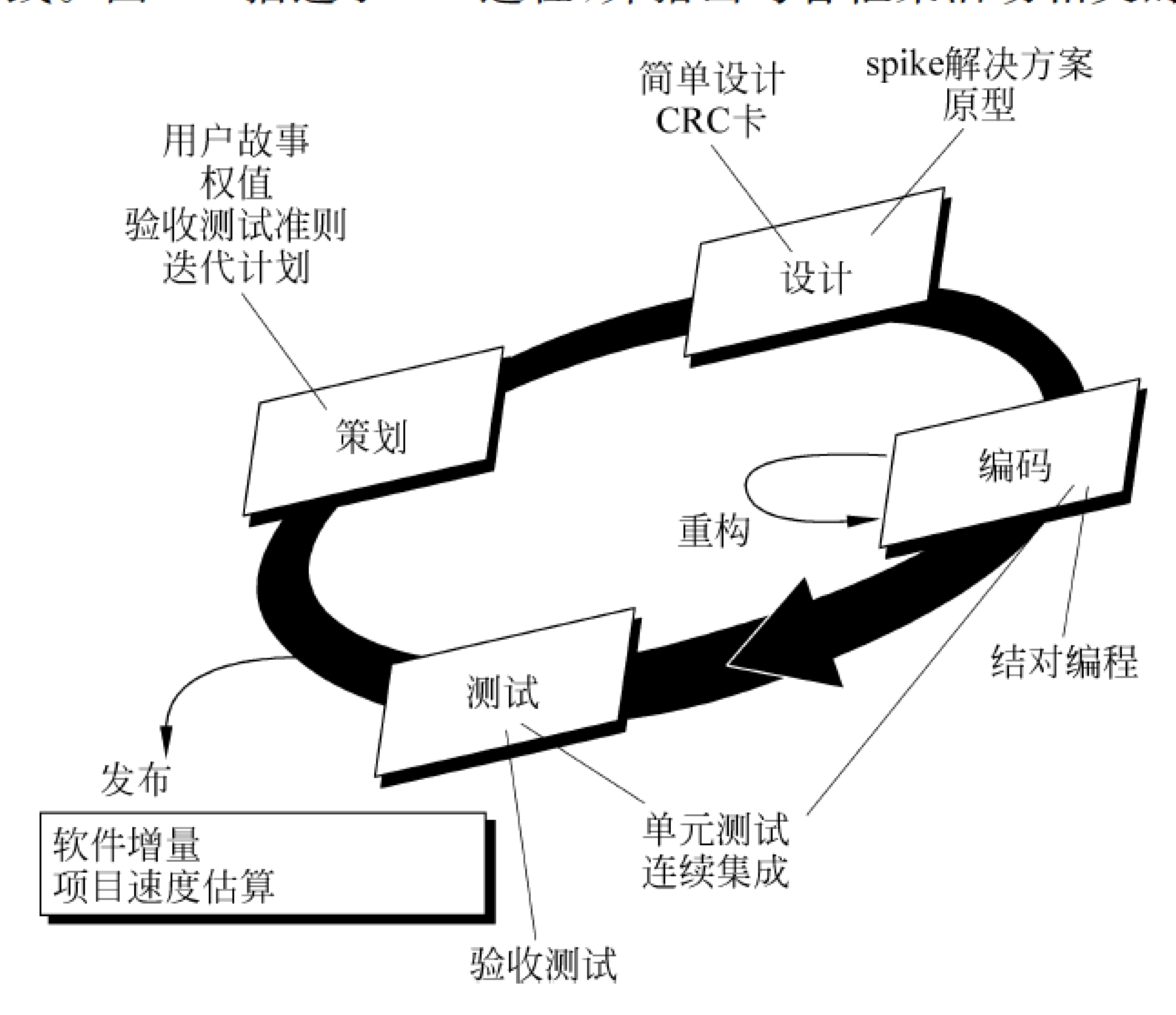


图 1-2　极限编程过程

1）策划

策划活动开始于建立一系列描述待开发软件的必要特征与功能要求。每个功能要求由客户书写并置于一张索引卡上，客户根据对应特征或功能的全局业务价值标明权值（即优先级）。XP团队成员评估每一个功能并给出以开发周数为度量单位的成本。如果某个功能的开发成本超过了3周，将请客户把该功能进一步细分，重新赋予权值并计算成本。新的功能要求可以在任何时刻书写。

客户和XP团队共同决定如何把功能分组置于XP团队将要开发的下一个发行版本中。一旦形成关于一个发布版本的基本承诺，XP团队将按以下三种方式之一对待开发的功能进行排序：

① 所有选定功能将在几周之内尽快实现；

② 具有最高价值的功能将移到进度表的前面并首先实现；

③ 高风险功能将首先实现。

项目的第一个发行版本发布之后，XP团队计算项目的速度。简言之，项目速度是第一个发行版本中实现的用户功能个数。项目速度将用于帮助建立后续发行版本的发布日期和进度安排；确定是否对整个开发项目中的所有功能有过分承诺。一旦发生过分承诺，则调整软件发行版本的内容或者改变最终交付日期。

在开发过程中，客户可以增加功能、改变功能要求的权值、分解或者去掉功能。接下来由XP团队重新考虑所有剩余的发行版本，并相应修改计划。

2）设计

XP设计严格遵循保持简洁（Keep It Simple，KIS）原则，使用简单而不是复杂的表述。另外，设计为功能提供不多也不少的实现原则，不鼓励额外功能性设计。

XP鼓励使用类—责任—协作者（CRC）卡作为有效机制，在面向对象环境中考虑软件、CRC卡的确定，组织和当前软件增量相关的对象和类。CRC卡也是作为XP过程一部分的唯一的设计工作产品。

如果在某个功能设计中碰到困难，XP推荐立即建立这部分设计的可执行原型，实现并评估设计原型，目的是在真正的实现开始时降低风险，对可能存在设计问题的功能确认最初的估计。

3）编码

在功能开发和基本设计完成之后，团队不应直接开始编码，而是开发一系列用于检测本次（软件增量）发布的包括所有功能的单元测试。一旦建立起单元测试，开发者就可以更集中精力于必须实现的内容以通过单元测试。不需要加任何额外的东西（保持简洁）。一旦编码完成，就可以立即完成单元测试，并向开发者提供即时的反馈。

XP编码活动中的关键概念之一是结对编程。XP推荐两个人面对同一台计算机共同为一个功能开发代码。这一方案提供实时解决问题和实时质量保证的机制，同时也使开发者能集中精力于手头的问题。实施中不同成员担任的角色略有不同。例如，一名成员考虑特定设计的详细编码实现，而另一名成员确保编码遵循特定的标准，生成的代码符合该功能要求的接口设计。

结对的两人完成所开发代码和其他工作集成。在有些情况下，这种集成工作由集成团队按日实施。在另外一些情况下，结对者自己负责集成，这种“连续集成”策略有助于避免兼容性和接口问题，建立能及早发现错误的“冒烟测试”环境。

4）测试

在编码开始之前建立单元测试是XP方法的关键因素。所建立的单元测试应当使用一个

可以自动实施的框架，这种方式支持代码修改之后即时的回归测试策略。

一旦将个人的单元测试组织到一个"通用测试集"，每天都可以进行系统的集成和确认测试。这可以为 XP 团队提供连续的进展指示，也可在一旦发生问题的时候及早提出预警。

XP 验收测试也称为客户测试，由客户确定，将着眼于客户可见的、可评审的系统级的特征和功能，验收测试根据本次软件发布中所实现的用户功能而确定。

习题

1. 简述软件的定义，软件具有什么样的特性。
2. 什么是软件危机？产生软件危机的原因是什么？如何消除？
3. 什么是软件工程？什么是软件生命周期？它们都包含哪些内容？
4. 软件工程都涉及哪些概念和名词？它们的关系如何？如何解释？
5. 运用软件工程的理论、技术和方法能够解决什么问题？
6. 如何理解软件开发工具和软件工程环境在软件工程中的作用？
7. 软件项目管理涉及哪些方面，它的必要性是什么？
8. 软件复审的目的是什么，怎样进行技术复审？软件技术审查和管理复审的作用是什么？
9. 什么是软件开发模型？软件开发模型有几种？各有什么特点？
10. 什么是敏捷开发？敏捷开发都有哪些方法？其基本过程是怎样的？

第2章 软件测试基本概念

信息技术的飞速发展，使软件产品应用到社会的各个领域，软件产品的质量自然成为人们共同关注的焦点。不论软件的生产者还是软件的使用者，均生存在竞争的环境中，软件开发商为了占有市场，必须把产品质量作为企业的重要目标之一，以免在激烈的竞争中被淘汰出局。用户为了保证自己业务的顺利完成，当然希望选用优质的软件。质量不佳的软件产品不仅会使开发商的维护费用和用户的使用成本大幅增加，还可能产生其他的责任风险，造成公司信誉下降，继而冲击股票市场。在一些关键应用（如民航订票系统、银行结算系统、证券交易系统、自动飞行控制系统、军事防御和核电站安全控制系统等）中使用质量有问题的软件，还可能造成灾难性的后果。

事实上，对于软件来讲，还没有像“银弹”那样的特效武器。在开发软件过程中，不论采用什么技术和什么方法，软件开发者都难免在工作中犯错误，从而使软件产品隐藏着许多错误和缺陷，规模大、复杂性高的软件更是如此。采用高级的语言、先进的开发方式、完善的开发过程，虽然可以减少错误的引入，但是不可能完全杜绝软件中的错误。这些错误有些是致命性的，如不排除，就会导致生命与财产的重大损失。这些错误需要通过测试来找出，软件中的错误密度也需要通过测试来进行估计。

2.1 软件测试概述

测试是所有工程学科的基本组成单元，是软件开发的重要组成部分，是软件工程的重要分支。软件测试是确保软件质量的重要一环，测试是手段，质量是目的。自有程序设计的那天起测试就一直伴随着。统计表明，在典型的软件开发项目中，软件测试工作量往往占软件开发总工作量的40%以上。而在软件开发的总成本中，用在测试上的开销要占30%～50%。如果把维护阶段也考虑在内，讨论整个软件生命周期时，测试的成本比例也许会有所降低，但实际上维护工作相当于二次开发，乃至多次开发，其中必定还包含有许多测试工作。因此，测试对于软件生产来说是必需的，问题是我们应该思考“采用什么方法、如何安排测试”。

2.1.1 软件测试发展史

软件测试是伴随着软件的产生而产生的。

1. 测试等同于“调试”

早期的软件开发过程中，那时软件规模都很小、复杂程度低，软件开发的过程混乱无序、相当随意，测试的含义比较狭窄，开发人员将测试等同于“调试”，目的是纠正软件中已经知道的故障，常常由开发人员自己完成这部分的工作。对测试的投入极少，测试介入也晚，常常是等到形成代码，产品已经基本完成时才进行测试。

直到 1957 年，软件测试才开始与调试区别开来，作为一种发现软件缺陷的活动。

2. 测试是一种发现软件缺陷的活动

由于一直存在着“为了让我们看到产品在工作，就得将测试工作往后推一点”的思想，潜意识里对测试的目的就理解为“使自己确信产品能工作”。测试活动始终后于开发的活动，测试通常被作为软件生命周期中最后一项活动而进行。当时也缺乏有效的测试方法，主要依靠“错误推测(Error Guessing)”来寻找软件中的缺陷。因此，大量软件交付后，仍存在很多问题，软件产品的质量无法保证。

因此，我们可以说，在软件工程建立之前的 20 世纪 60 年代，软件测试是为表明程序正确而进行的测试。

到了 20 世纪 70 年代，这个阶段开发的软件仍然不复杂，但人们已开始思考软件开发流程的问题，尽管对“软件测试”的真正含义还缺乏共识，但这一词条已经频繁出现，一些软件测试的探索者们建议在软件生命周期的开始阶段就根据需求制定测试计划，这时也涌现出一批软件测试的宗师，Bill Hetzel 博士就是其中的一位。

1972 年，软件测试领域的先驱 Bill Hetzel 博士(代表论著 *The Complete Guide to Software Testing*)，在美国的北卡罗来纳(North Carllina)大学组织了历史上第一次正式的关于软件测试的会议。

1973 年，Bill Hetzel 博士给软件测试一个这样的定义：“就是建立一种信心，认为程序能够按预期的设想运行(Establish confidence that a program does what it is supposed to do)”。

1975 年 John Good Enough 和 Susan Gerhart 在 IEEE 上发表了《测试数据选择的原理》的文章，首次提出了软件测试理论。同年 Huang 全面讨论了测试过程和测试准则，从此软件测试被确定为一种研究方向。

1979 年，Glenford Myers 在 *The Art of Software Testing*(《软件测试艺术》)一书中提出测试的目的是证伪，即测试是为发现错误而执行的一个程序或者系统的过程。

3. 现代软件测试定义的产生

20 世纪 80 年代早期，“质量”的号角开始吹响，各个软件企业开始建立 QA/SQA 部门及其演化流程，软件测试定义发生了改变，测试不单纯是一个发现错误的过程，而且包含软件质量评价的内容。制定了各类标准。

1981 年，Bill Hetzel 博士开设了一门公共课“结构化软件测试”(Structured Software Testing)。1983 年，他在出版的《软件测试完全指南》一书中对 1973 年他所给出的软件测试定义进行了修订：“测试是评价一个程序或者系统属性及其能力的各种活动(Any activities aimed at evaluating an attribute or capability of a program or system)”。

1988 年，David Gelperin 和 Bill Hetzel 在《Communications of the ACM》发表《软件测试进展》(The Growth of Software Testing)论文，介绍系统化的测试和评估流程。

80 年代后期，Paul Rook 提出了著名的软件测试的 V 模型，旨在改进软件开发的效率和效果。从此，软件测试模型与软件测试标准的研究也随着软件工程的发展而越来越深入。

90 年代，测试工具盛行起来。

1996 年提出了测试能力成熟度模型(Testing Capability Maturity Model，TCMM)、测试支持度模型(Testability Support Model，TSM)、测试成熟度模型(Testing Maturity Model，TMM)。

到了 2002 年，Rick 和 Stefan 在《系统的软件测试》一书中对软件测试做了进一步定义：测试是为了度量和提高被测软件的质量，对测试软件进行工程设计、实施和维护的整个生命周

期过程。

2.1.2 软件测试定义

1. Bill Hetzel 博士的软件测试定义

软件测试的定义最早是 Bill Hetzel 博士在 1973 年提出来的。他认为软件测试的目的是建立一种信心,认为程序能够按预期的设想运行。后来在 1983 年他又将定义修订为:评价一个程序和系统的特性或能力,并确定它是否达到预期的结果。软件测试就是以此为目的的各种行为。在他的定义中的“预期的结果”,其实就是我们现在所说的用户需求或功能设计。他还把软件的质量定义为“符合要求”。Bill Hetzel 博士思想的核心观点是:测试方法是试图验证软件是“工作的”,即指软件的功能是按照预先的设计执行的,以正向思维,针对软件系统的所有功能点,逐个验证其正确性。软件测试业界把这种方法看做软件测试的第一类方法。

2. Glenford Myers 的软件测试定义

尽管如此,Bill Hetzel 这一方法还是受到很多业界权威的质疑和挑战。代表人物是前面提到的 Glenford Myers。他认为测试不应该着眼于验证软件是工作的,相反应该首先认定软件是有错误的,然后用逆向思维去发现尽可能多的错误。他还从人的心理学的角度论证,如果将“验证软件是工作的”作为测试的目的,非常不利于测试人员发现软件的错误。于是他在我们前面提到的 *The Art of Software Testing*(《软件测试艺术》)一书中提出了他对软件测试的定义:测试是为发现错误而执行一个程序或者系统的过程。这个定义,也被业界所认可,经常被引用。除此之外,Myers 还给出了与测试相关的 3 个重要观点,那就是:

① 测试是为了证明程序有错,而不是证明程序无错误;

② 一个好的测试用例在于它能发现至今未发现的错误;

③ 一个成功的测试是发现了至今未发现的错误的测试。

这就是软件测试的第二类方法,简单地说就是验证软件是“不工作的”,或者说是有错误的。Myers 认为,一个成功的测试必须是发现 Bug 的测试,不然就没有价值。这就如同一个病人(假定此人确有病),到医院做一项医疗检查,结果各项指标都正常,那说明该项医疗检查对于诊断该病人的病情是没有价值的,是失败的。

Myers 提出的“测试的目的是证伪”这一概念,推翻了过去“为表明软件正确而进行测试”的错误认识,为软件测试的发展指出了方向,软件测试的理论、方法在之后得到了长足的发展。第二类软件测试方法在业界也很流行,受到很多学术界专家的支持。

然而,对 Glenford Myers 先生“测试的目的是证伪”这一概念的理解也不能太过于片面化。在很多软件工程学、软件测试方面的书籍中都提到一个概念:“测试的目的是寻找错误,并且是尽最大可能找出最多的错误”。这很容易让人们认为测试人员就是“挑毛病”的,而由此带来诸多问题。包括大家熟悉的 Ron Patton 在《软件测试》一书中有一个明确而简洁的定义:软件测试人员的目标是尽可能早一些找到软件缺陷,并确保其得以修复。这样的定义具有一定的片面性,带来的结果是:①若测试人员以发现缺陷为唯一目标,而很少去关注系统对需求的实现,测试活动往往会存在一定的随意性和盲目性;②若有些软件企业接受了这样的方法,以 Bug 数量来作为考核测试人员业绩的唯一指标,也不太科学。

总的来说,第一类测试可以简单抽象地描述为这样的过程:在设计规定的环境下运行软件的功能,将其结果与用户需求或设计结果相比较,如果相符则测试通过,如果不相符则视为 Bug。这一过程的终极目标是将软件的所有功能在所有设计规定的环境全部运行,并通过。在软件行业中一般把第一类方法奉为主流和行业标准。第一类测试方法以需求和设计为本,

因此有利于界定测试工作的范畴,更便于部署测试的侧重点,加强针对性。这一点对于大型软件的测试,尤其是在有限的时间和人力资源情况下显得格外重要。

而第二类测试方法与需求和设计没有必然的关联,更强调测试人员发挥主观能动性,用逆向思维方式,不断思考开发人员理解的误区、不良的习惯、程序代码的边界、无效数据的输入以及系统各种的弱点,试图破坏系统、摧毁系统,目标就是发现系统中各种各样的问题。这种方法往往能够发现系统中存在的更多缺陷。

3. 现代软件测试定义

到了20世纪80年代初期,软件和IT行业进入了大发展,软件趋向大型化、复杂化,软件的质量越来越重要。这个时候,一些软件测试的基础理论和实用技术开始形成,并且人们开始为软件开发设计了各种流程和管理方法,软件开发的方式也逐渐由混乱无序的开发过程过渡到结构化的开发过程,以结构化分析与设计、结构化评审、结构化程序设计以及结构化测试为特征。人们还将"质量"的概念融入其中,软件测试定义发生了改变,测试不单纯是一个发现错误的过程,而且将测试作为软件质量保证(SQA)的主要职能,包含软件质量评价的内容。软件开发人员和测试人员开始坐在一起探讨软件工程和测试问题。软件测试出台了行业标准,1983年IEEE提出的软件工程术语中给软件测试下的定义是:"使用人工或自动的手段来运行或测定某个软件系统的过程,其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别"。这个定义明确指出:软件测试的目的是检验软件系统是否满足需求。它再也不是一个一次性的,而且只是开发后期的活动,而是与整个开发流程融合成一体。软件测试已成为一个专业,需要运用专门的方法和手段,需要专门人才和专家来承担。

事实上,人们目前对软件测试的认识从广义上讲,软件测试是指软件产品生命周期内所有的检查、评审和确认活动,如设计评审、系统测试;从狭义上讲,软件测试是对软件产品质量的检验和评价。它一方面检查软件产品质量中存在的质量问题,同时对产品质量进行客观的评价。基于这些认识,可以给出软件测试的定义:软件测试就是在软件投入运行前,对软件需求分析、设计规格说明和编码的最终复查,是软件质量保证的关键步骤。在该定义下,借鉴Bill Hetzel和Glenford Myers有关软件测试的思想,我们可以引出如下的重要概念:

(1) 软件测试是对程序或系统能否完成特定任务建立信心的过程,也是帮助识别开发完成(中间或最终的版本)的计算机软件(整体或部分)的正确性(Correctness)、完整性(Completeness)和质量(Quality)的软件过程。

(2) 软件测试就是为了发现程序中的错误而分析或执行程序的过程,或者说是根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例,并利用这些测试用例去运行程序,以发现程序错误的过程。

(3) 软件测试的目标在于尽可能地发现错误(缺陷)。

(4) 软件测试目的在于鉴定程序或系统的属性或能力的各种活动,是软件质量的一种度量,是软件质量保证(SQA)的重要子域。

(5) 使用人工或自动手段来运行或测定某个系统的过程,其目的在于检验它是否满足规定的需求(遗漏、超出)或是弄清预期结果与实际结果之间是否有差别。

软件测试在软件生命周期中横跨两个阶段。

(1) 单元测试阶段:编写出每个模块之后,就对它做必要的测试。

(2) 综合测试阶段:结束单元测试后进行的测试,如系统测试、验收测试等。

2.1.3 软件测试目的

用户普遍希望通过软件测试暴露软件中隐藏的错误和缺陷,以考虑是否可接受该产品;

软件开发者则希望测试成为表明软件产品中不存在错误的过程,验证该软件已正确地实现了用户的要求,确立人们对软件质量的信心。而软件测试者则是替用户受过。

早期人们做测试,所期望达到的目的有几点:①测试是程序的执行过程,目的在于发现错误;②一个好的测试用例在于能发现至今未发现的错误;③一个成功的测试是发现了至今未发现的错误的测试。

1. 当前关于软件测试目的的几种观点

下面介绍当前关于软件测试目的的几种观点。

(1) 软件测试的目的是尽可能发现并改正被测软件中的错误,提高被测软件的可靠性。

这个观点听起来很正确,但用它来指导测试会带来很多问题。比如有的组织用发现的bug数来衡量测试人员的业绩,其实这就是这种测试目的论在后面作祟,其结果:①有一些不够敬业的测试人员会找来一些无关痛痒的bug来充数,结果许多时间会被浪费在这些无关痛痒的bug上;②测试人员会花很大精力设计一些复杂的测试用例去发现一些迄今尚未发现的缺陷,而不关心这些缺陷在实际用户的使用过程当中是否会发生,从而浪费了大量的宝贵时间。究其根源,就是因为对测试目的的这种错误理解造成的,为什么这么说呢?因为软件里bug的数量是无从估计的,那么如果测试的目的是找bug,那么测试工作将变成一项无法完成也无法衡量进度而且部分无效的工作(因为有些bug在实际的运行过程当中根本不会发生)。

(2) 软件测试的目的就是保证软件质量。

这个观点也是看似正确,但实际上,混淆了测试和质量保证工作的边界。软件质量要素有很多,包括可理解性、简洁性、可移植性、一致性、可维护性、可测试性、可用性、有效性、安全性等,所以,软件质量保证和测试其实关注的方向是不同的。

(3) IEEE观点。

实际上,正确的软件测试目的概念在IEEE 1983年提出的软件测试定义中明确给定:使用人工或自动手段来运行或测定某个系统的过程,其目的在于检验它是否满足规定的需求或者弄清预期结果与实际结果之间的差别。所以,软件测试的目的应该是验证需求,bug(预期结果与实际结果之间的差别)是这个过程中的产品而非目标。测试人员在软件产品投入使用之前对该需求进行测试检验或验证,把尽可能多的问题在产品交给用户之前发现并改正。

总之,测试的目的是系统地找出软件中潜在的各种错误和缺陷,并能够证明软件的功能和性能与需求说明相符合。要注意的是:测试不能表明软件中不存在错误,它只能说明软件中存在错误。

2. 软件测试一般要达到的具体目标

软件测试一般要达到如下具体目标。

(1) 确保产品完成了它所承诺或公布的功能,并且所有用户可以访问到的功能都有明确的书面说明。

产品缺少明确的书面文档,是厂商一种短期行为的表现,也是一种不负责任的表现。所谓短期行为,是指缺少明确的书面文档既不利于产品最后的顺利交付,容易与用户发生矛盾,影响厂商的声誉和将来与用户的合作关系;同时也不利于产品的后期维护,也使厂商支出超额的用户培训和技术支持费用。从长期利益看,这是很不划算的。

当然,书面文档的编写和维护工作对于使用快速原型法(RAD)开发的项目是最为重要的、最为困难,也是最容易被忽略的。

最后,书面文档的不健全甚至不正确,也是测试工作中遇到的最大和最头痛的问题,它的

直接后果是测试效率低下、测试目标不明确、测试范围不充分，从而导致最终测试的作用不能充分发挥、测试效果不理想。

(2) 确保产品满足性能和效率的要求。

系统使用起来其运行效率低(性能低)，或用户界面不友好、用户操作不方便(效率低)的产品不能说是一个有竞争力的产品。实际上，用户最关心的不是你的技术有多先进、功能有多强大，而是他能从这些技术、这些功能中得到多少好处。也就是说，用户关心的是他能从中取出多少，而不是你已经放进去多少。

(3) 确保产品是健壮的和适应用户环境的。

健壮性即稳定性，是产品质量的基本要求，尤其对于一个用于事务关键或时间关键的工作环境中的应用系统。软件只有稳定地运行，才不至于中断用户的工作，因此通过健壮性测试是软件测试工作的又一目标。

另外就是不能假设用户的环境(某些项目可能除外)。

2.1.4 软件测试原则

软件测试经过几十年的发展，测试界提出了很多软件测试的基本原则，为测试管理人员和测试人员提供了测试指南。软件测试原则非常重要，测试人员应该在测试原则指导下进行测试活动。另外，软件测试的基本原则有助于测试人员进行高质量的测试，尽早尽可能多地发现缺陷，并负责跟踪和分析软件中的问题，对存在的问题和不足提出质疑和改进，从而持续改进测试过程。

很显然，对于相对复杂的产品或系统而言，zero-bug(没有错误)是我们的理想目标，good-enough(足够好)是我们的工作原则。

1. 足够好原则

good-enough 原则就是一种权衡投入/产出比的原则：不充分的测试是不负责任的；过分的测试是一种资源的浪费，同样也是一种不负责任的表现。我们的操作困难在于：如何界定什么样的测试是不充分的，什么样的测试是过分的。目前状况唯一可用的答案是：制定最低测试通过标准和测试内容，然后具体问题具体分析。

2. 木桶原理和 80-20 原则

在确定软件测试原则时，我们一定要注意软件测试的规律——木桶原理和 80-20 原则。

1) 木桶原理

所谓“木桶理论”也即“木桶定律”，其核心内容为：一只木桶盛水的多少，并不取决于桶壁上最高的那块木块，而恰恰取决于桶壁上最短的那块。根据这一内容，可以有两个推论：

① 只有桶壁上的所有木板都足够高，那木桶才能盛满水；

② 只要这个木桶里有一块不够高度，木桶里的水就不可能是满的。

在软件产品生产方面就是全面质量管理(TQM)的概念。产品质量的关键因素是分析、设计和实现，测试应该是融于其中的补充检查手段，其他管理、支持甚至文化因素也会影响最终产品的质量。应该说，测试是提高产品质量的必要条件，也是提高产品质量最直接、最快捷的手段，但绝不是一种根本手段。反过来说，如果将提高产品质量的砝码全部押在测试上，那将是一个恐怖而漫长的灾难。

2) Bug 的 80-20 原则

80%的软件缺陷常常存在于软件 20%的代码中。这个原则告诉我们，如果你想使软件测试有效的话，就要关注高危多发的代码。在那里发现软件缺陷的可能性会大得多，效果也会好

得多。这一原则对于软件测试人员提高测试效率及缺陷发现率有着重大的意义。

80-20原则的另外一种情况是：在分析、设计、实现阶段的复审和测试工作能够发现和避免80%的Bug，而系统测试又能找出其余Bug中的80%，最后的4%的Bug可能只有在用户的大范围、长时间使用后才会暴露出来。因为测试只能够保证尽可能多地发现错误，无法保证能够发现所有的错误。

3. 软件测试的一般原则

软件测试的一般原则如下。

(1) 测试应该基于用户需求，应该基于“质量第一”的要求去开展工作。

依照用户的应用要求、配置环境和使用习惯进行测试并评价结果。软件测试只能证明软件中存在错误，而不能表明软件中没有错误。软件测试所起的作用也就是用于确定程序中缺陷的存在并帮助人们判断程序在实际中是否可用。而软件或程序是否可用必须依据事先定义好的产品质量标准。

(2) 项目一启动，软件测试就开始，而不是等程序写完才开始测试。

应该尽早开始测试，如尽早制定测试计划，尽早进行测试设计，以及测试从模块级开始等。其中，测试设计是关键，因为测试时间和资源是有限的，测试所有情况是不可能的，但要避免冗余的测试；另外，软件测试中最困难的问题之一是何时停止测试，何时完成测试任务。

(3) 充分覆盖程序逻辑，而第三方测试会更有效，更客观。

在软件测试中最忌讳的是程序编写者测试自己编写的程序，因为他们很难以批判者的姿态对自己编写的程序挑毛病。另外，他们的逻辑思维已定型，难以发现逻辑上的问题。

4. 基于经验的软件测试具体原则

我们在总结软件测试经验的基础上给出如下软件测试的具体原则。

1) 测试工作可以发现和显示软件缺陷的存在，但不能证明软件或系统不存在缺陷

测试可以减少软件中存在缺陷的可能性，但即使测试没有发现任何缺陷，也不能证明软件或系统是完全正确的，或者说是不存在缺陷的。

2) 测试的尽早介入

根据统计表明，在软件生命周期早期引入的错误占软件过程中出现的所有错误(包括最终的缺陷)数量的50%～60%。此外，IBM的一份研究结果表明，缺陷存在放大趋势。如需求阶段的一个错误可能会导致N个设计错误，因此，越是测试后期，为修复缺陷所付出的代价就会越大。因此，软件测试人员要尽早地且不断地进行软件测试，以提高软件质量，降低软件开发成本。

3) 测试活动要有组织、有计划、有选择

把“尽早地和不断地进行软件测试”作为软件开发者和软件测试者的座右铭；同时要明确测试工作量——穷举测试是不可能的，测试太少是不负责任，测试过多是一种浪费。因此，测试中有计划的活动能够大大地提高测试效率。

4) 选择最佳的测试策略

100%的测试是不可能的，不同的测试组织采用的测试策略是不同的，但我们一定要考虑软件测试的多、快、好、省效果。

所谓多、快、好、省就是：

① 能够找到尽可能多的、以至于所有的bug。

② 能够尽可能早地发现最严重的bug。

③ 找到的 bug 是关键的、用户最关心的，找到 bug 后能够重现找到的 bug，并为修正 bug 提供尽可能多的信息。

④ 能够用最少的时间、人力和资源发现 bug，测试的过程和数据可以重用。

5）注重测试设计

测试设计决定了测试的有效性和效率，测试工具只能提高测试效率，测试用例设计则是软件测试的关键。

测试用例应由测试输入数据和对应的预期输出结果组成。设计测试用例时，应包括合理的输入条件和不合理的输入条件。一个好的测试用例应当是一个对以前未被发现的缺陷有高发现率的用例，而不是一个表明程序工作正确的用例。

另外，测试用例需要经常地评审和修改，不断增加新的不同的测试用例来测试软件或系统的不同部分，保证测试用例永远是最新的，即包含着最后一次程序代码或说明文档的更新信息。这样软件中未被测试过的部分或者先前没有被使用过的输入组合就会重新执行，从而发现更多的缺陷。

由于软件测试的不成熟性和艺术性，我们在软件测试中不要放弃随机测试的方法。

6）严格执行测试计划

测试中，一定要严格执行测试计划，坚决排除测试的随意性，必须对每一个测试结果做全面检查。

在测试中，程序员应避免检查自己的程序，否则很难发现思路错误和环境错误，或因心理因素导致测试可能不够彻底和全面。

测试人员要充分注意测试中因被测程序的编码规范、需求理解、技术能力、内部耦合性等导致错误扎堆这种“虫子窝”现象。一般测试后程序中残存的错误数目与该程序中已发现的错误数目成正比，当一个软件被测出的错误数目增加时，更多的未被发现的错误存在的概率也随之增加。

另外，测试前必须明确预期的输出结果，否则实际的输出结果很可能成为检验的标准，测试失去意义。同时，作为专业的测试人员，要具有探索性思维和逆向思维，而不仅仅是做输出与期望结果的比较。

7）测试活动依赖于测试内容

项目测试相关的活动依赖于测试对象的内容。对于每个软件系统，测试策略、测试技术、测试工具、测试阶段以及测试出口准则等的选择都是不一样的。同时，测试活动必须与应用程序的运行环境和使用中可能存在的风险相关联。因此，没有两个系统可以以完全相同的方式进行测试。比如，对关注安全的电子商务系统进行测试，与一般的商业软件测试的重点是不一样的，它更多关注的是安全测试和性能测试。

8）没有失效不代表系统是可用的

系统的质量特征不仅仅是功能性要求，还包括了很多其他方面的要求，如稳定性、可用性、兼容性等。假如系统无法使用，或者系统不能满足客户的需求和期望，那么，这个系统的研发是失败的。同时在系统中发现和修改缺陷也是没有任何意义的。

在开发过程中用户的早期介入和接触原型系统就是为了避免这类问题的预防性措施。有时候，可能产品的测试结果非常完美，可最终的客户并不买账。因为，这个开发完美的产品可能并不是客户真正想要的产品。

9）测试的标准是用户的需求

提供软件的目的是帮助用户完成预定的任务，并满足用户的需求。这里的用户并不特指

最终软件测试使用者。比如我们可以认为系统测试人员是系统需求分析和设计的客户。软件测试的最重要的目的之一是发现缺陷,因此测试人员应该在不同的测试阶段站在不同用户的角度去看问题,系统中最严重的问题是那些无法满足用户需求的错误。

10) 尽早定义产品的质量标准

只有建立了质量标准,才能根据测试的结果,对产品的质量进行分析和评估。同样,测试用例应该确定期望输出结果。如果无法确定测试期望结果,则无法进行检验。必须用预先精确对应的输入数据和输出结果来对照检查当前的输出结果是否正确,做到有的放矢。

11) 测试贯穿于整个生命周期

由于软件的复杂性和抽象性,在软件生命周期的各个阶段都可能产生错误,测试的准备和设计必须在编码之前就开始,同时为了保证最终的质量,必须在开发过程的每个阶段都保证其过程产品的质量。因此不应当把软件测试仅仅看做软件开发完成后的一个独立阶段的工作,应当将测试贯穿于整个生命周期始末。

软件项目一启动,软件测试就应该介入,而不是等到软件开发完成。在项目启动后,测试人员在每个阶段都应该参与相应的活动。或者说每个开发阶段,测试都应该对本阶段的输出进行检查和验证。比如在需求阶段,测试人员需要参与需求文档的评审。

12) 第三方或独立的测试团队

由于心理因素,人们潜意识都不希望找到自己的错误。基于这种思维定势,人们难于发现自己的错误。因此,由严格的独立测试部门或者第三方测试机构进行软件测试将更客观、公正,测试活动也会达到更好效果。

但是,第三方或者独立的测试团队这个原则,并不是认为所有的测试完全由他们来完成。一定程度的独立测试,可以更加高效地发现软件缺陷和软件存在的问题。但独立测试也不是绝对的,因为开发人员也可以高效地在他们的代码中找出很多缺陷。在软件开发的早期,开发人员对自己的工作产品进行认真测试,这也是开发人员的一个职责之一。

13) 妥善保存测试计划、测试用例、出错统计和最终分析报告

通过配置管理、测试管理等技术手段,保存测试计划、测试用例、出错统计和最终分析报告,除了方便维护及结果重现外,对于组织和团队的建设、工作的改进、成果的积累、资源的重用等有重大好处。

2.1.5 软件测试质量度量

软件测试是软件质量控制的重要方式和重要手段,但软件测试本身的质量质量又该如何度量?尽管有关这个问题目前还没有权威的结论,但也有一些共识,如:软件测试度量的难度在于不能直接从软件产品的质量反应软件测试的效果。对于软件测试的度量,应该从对软件产品的度量转移到软件测试产出物的度量,以及测试过程的度量。

软件测试质量度量的目的是改进软件测试的质量,提高测试效率,改进测试过程的有效性。开展软件测试质量度量,最关键的一项工作就是对软件测试人员的工作质量度量。这是因为测试人员是测试过程的核心人物,测试人员的工作质量会极大地影响测试的质量以及产品的质量。对测试人员的工作做出评价一般由测试经理或项目经理、质量保证人员以及开发人员这三类人员进行综合考核或评判。除了人员考核这个管理内容外,表 2-1 给出了软件开发与软件测试质量度量的相关指标及度量范围。

表 2-1　软件测试的质量度量

指标名称	定　　义	度量范围
工作量偏差	((实际工作量－计算工作量)/计划工作量)×100%	进度
测试执行率	(实际执行的测试用例数/测试用例总数)×100%	测试进度
测试通过率	(执行通过的测试用例数/测试用例总数)×100%	开发质量
需求(测试用例)覆盖率	(已设计测试用例的需求数/需求总数)×100%	测试设计质量
需求通过率	(已测试通过的需求数/需求总数)×100%	进度
测试用例命中率	(缺陷总数/测试用例数)×100%	测试用例质量
二次故障率	(Reopen 的缺陷/缺陷总数)×100%	开发质量
测试未通过率	(验证不通过的缺陷/缺陷总数)×100%	开发质量
缺陷有效率	(无效的缺陷/缺陷总数)×100%	测试
缺陷修复率	(已解决的缺陷/缺陷总数)×100%	开发
缺陷生存周期	缺陷从提交到关闭的平均时间	开发、测试
缺陷修复的平均时长	缺陷从提交到修复的平均时间	开发
缺陷关闭的平均时长	缺陷从修复到关闭的平均时间	测试
缺陷探测率	(测试者发现的缺陷数/(测试者发现的缺陷＋客户发现的缺陷))×100%	测试质量

2.1.6　软件测试与软件开发各阶段的关系

软件开发过程是一个自顶向下，逐步细化的过程。软件计划阶段定义软件作用域；软件需求分析建立软件信息域、功能和性能需求、约束等；然后进入软件开发，进行软件设计，进行编码，把设计用某种程序设计语言转换成程序代码。

测试过程是依相反顺序自底向上，逐步集成的过程。对每个程序模块进行单元测试，消除程序模块内部逻辑和功能上的错误和缺陷；对照软件设计进行集成测试、检测和排除子系统或系统结构上的错误；对照需求，进行确认测试；最后从系统全体出发，运行系统，看是否满足要求。软件测试与软件开发各阶段的关系参见图 2-1。

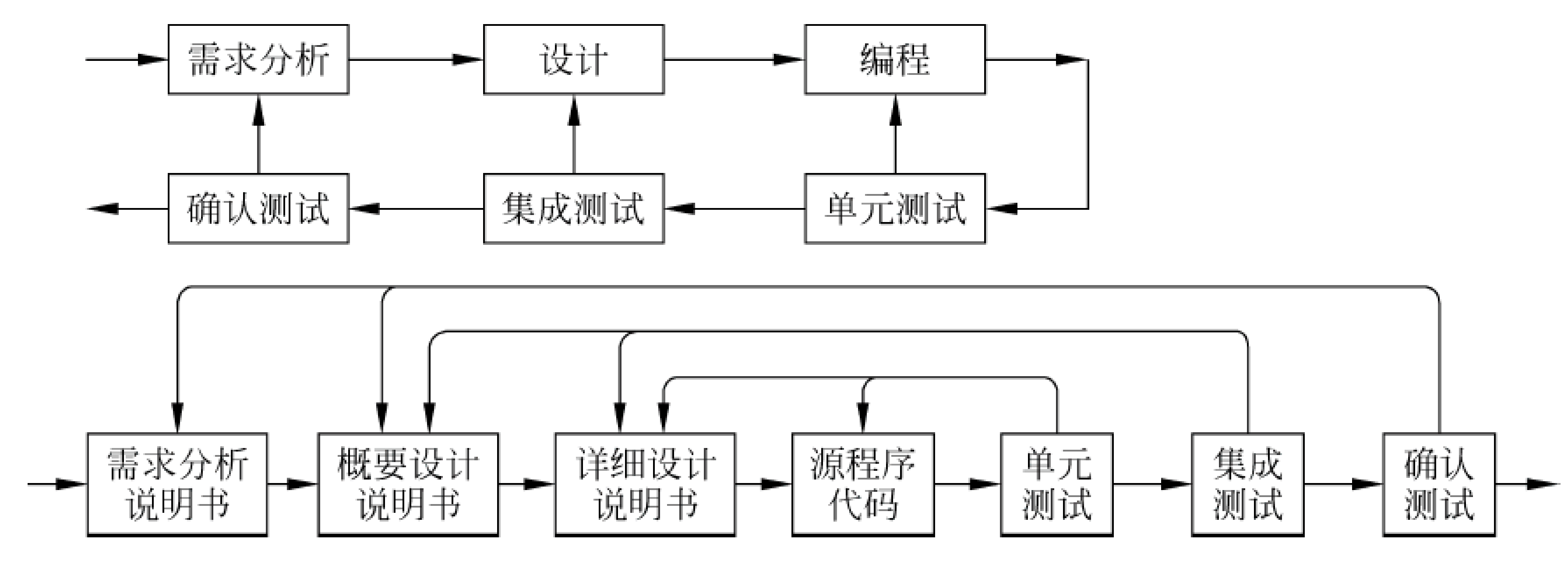


图 2-1　软件测试与软件开发各阶段的关系

软件测试是用人工或自动的方法执行软件并把观察到的行为特性与所期望的行为特性进行比较的过程。按照传统的观点，软件测试是软件开发过程中的一项活动。随着对软件测试方法、测试工具和测试技术的研究，测试的概念已经从编程后的评估过程发展成软件生命周期中每个阶段的一个必需的活动。软件开发对应的软件测试过程如图 2-2 所示。

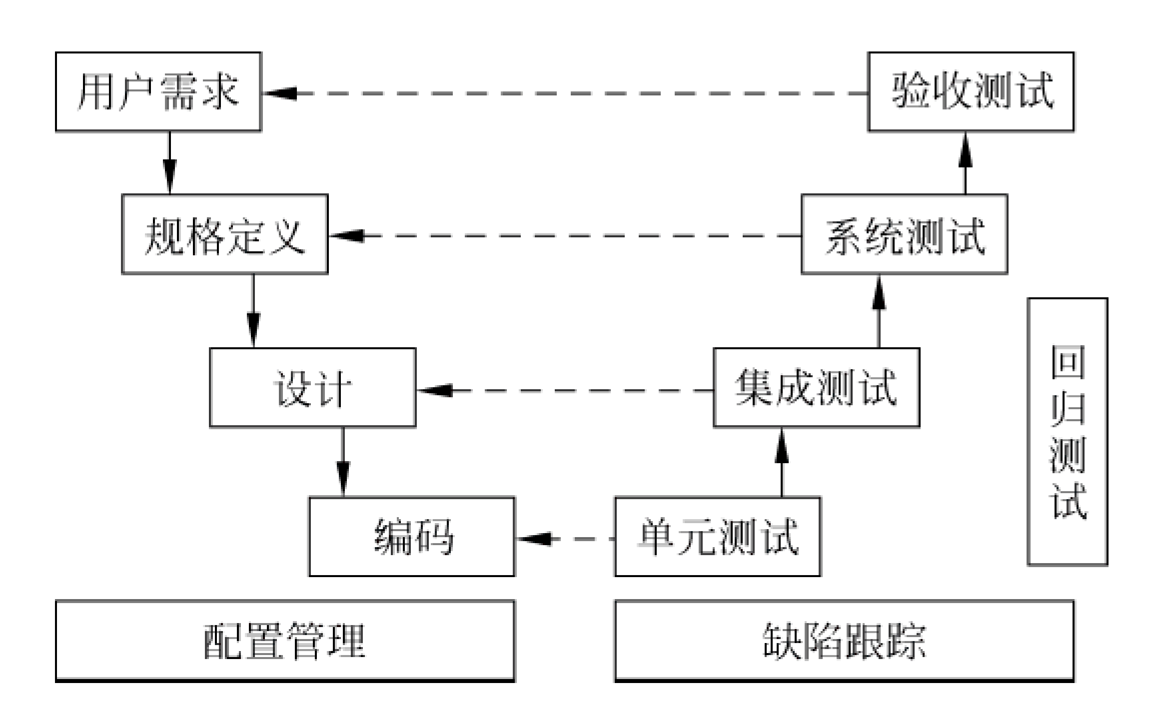


图 2-2 软件开发对应的软件测试过程

2.2 软件测试工作

软件开发工作的根本是尽量实现软件用户的需求，软件测试工作的根本是检验软件系统是否满足软件用户的需求。因此，我们可以说：软件测试主要工作内容是验证和确认软件的设计、开发是否符合需求。验证是保证软件正确地实现了客户的业务功能，即保证软件实现了需求所规定的内容；确认是通过使用已开发好的系统进行体验各项流程，目的是想证实在一个给定的外部环境中软件的逻辑是否正确。

不论是对软件模块还是整个系统，它们有着共同的测试内容，如正确性测试、容错性测试、性能与效率测试、可用性测试、文档测试等。常用的测试技术或手段是“白盒测试”及“黑盒测试”。

软件测试工作涉及多方面的内容和因素，但最为重要的是：测试工作的流程、测试工具的支持以及测试人员的素质这三项内容。

2.2.1 软件测试工作流程

软件测试工作贯穿整个软件开发周期，从需求提出到产品发布，测试人员几乎要全程参与其中，并按照特定的测试工作流程和企业对软件的质量要求开展测试工作。

1. 测试工作的基本流程

测试工作的基本流程如下。

1）需求阅读与评审

需求文档由 SE(System Engineer)根据客户需求编制，完成后提交到软件配置管理中的开发库。测试人员从开发库中下载该文档阅读，在阅读过程中要记录需求中模糊的部分、提出测试环境方案并绘制测试系统结构图、根据现有的软硬件条件对项目的可测性进行评估。

接下来是参加由 SE 发起的需求评审会议，在会议上测试人员的工作包括：从 SE 处获取需求文档中模糊部分的详细解释；向与会人员介绍测试环境方案，与会人员进行评审，评审完成后确定一个最终的方案；如果现有的条件不足以保证软件的各个模块都可测，需要就不可测的部分与参会人员探讨，最终提出一个解决方法，硬件不满足的购置设备，软件不满足的编制模拟程序。

2）用例设计与评审

在这个过程中，不要求每个用例都非常详细，但是要求用例要全面并充分考虑异常，保证各个模块都能够得到充分测试。

3）环境搭建

根据之前评审确定的测试环境方案搭建测试环境。

4）软件测试

从开发库中获取软件开发人员提交的软件测试版本，根据需求文档和之前设计的测试用例进行测试。如果发现问题，用缺陷管理工具提交问题单，问题单提交后，相关的开发人员会收到邮件提醒并解决问题，问题处理完毕后重新提交版本，测试人员重新获取版本并验证问题的解决情况，循环上述过程，直到整个软件测试完毕。

在此过程中，还需要细化测试用例，详细描述每一个用例的步骤。此外，每天工作结束后要向测试经理汇报测试进度，包括发现了哪些 bug，每个 bug 的处理情况（已提交问题单、开发人员已处理完毕、回归测试中、回归测试完毕等）。

5）编写有关测试文档（测试用例设计、测试报告、问题报告等）

测试用例可使用相关工具导出，形成 Word 文档；测试报告的内容包括测试环境结构图、测试环境介绍、测试方法、测试结论等。

6）SE 与测试经理审核

SE 和测试经理根据测试报告对测试工作进行审核，审核通过后测试工作完成。

2. 企业对软件质量的要求是测试工作开展的基础

在开展测试工作之前，企业必须做到对软件质量水平有明确的定位，坚持质量第一的原则，以客户满意为最终目标，要求测试人员发现任何问题都要及时反馈。同时规定：与需求的符合程度是衡量软件质量的唯一标准。以上两点使得测试人员在发现问题时有明确的处理原则和参照文档，避免了测试的随意性。这样，测试人员就能够充分地发挥他们的主观能动性，高质量地开展测试工作。如：

（1）测试人员在每一个项目的测试之初，认真地规划好时间点。从测试方案设计到环境搭建到版本测试，都有较为合理和详细的时间规定，保证测试在时间上的可控性。同时采用每天上报的方法，对测试进度进行跟踪，这对保证版本按时完成测试有重要意义。

（2）测试方案是否可行、是否能够对软件进行有效测试、缺少哪些软硬件设备，这些都需要进行充分论证，若有问题及时解决，直到整个方案符合要求。

（3）在测试用例评审时，对测试人员设计的用例进行讨论，补充遗漏，提示重点，特别是在异常的考虑上，能够集思广益，使得最终敲定的测试用例有良好的质量保证。

（4）测试人员注重自身专业技能的提高，保证测试工作的质量和效率。

2.2.2 软件测试工具对测试工作的支持

进入 20 世纪 90 年代，软件行业开始迅猛发展，软件的规模变得非常大，在一些大型软件开发过程中，测试活动需要花费大量的时间和成本，而当时测试的手段几乎完全都是手工测试，测试的效率非常低；并且随着软件复杂度的提高，出现了很多通过手工方式无法完成测试的情况，尽管在一些大型软件的开发过程中，人们尝试编写了一些小程序来辅助测试，但是这还是不能满足大多数软件项目测试的统一需要。于是，很多测试实践者开始尝试开发测试工具来支持测试，辅助测试人员完成某一类型或某一领域内的测试工作，而测试工具逐渐盛行起来。人们普遍意识到，工具不仅仅是有用的，而且要对今天的软件系统进行充分的测试，工具是必不可少的。

事实上，在繁重的测试工作中，测试人员使用软件测试工具有如下好处。

1. 提高工作效率

软件测试的工作量很大（据统计，测试会占用到 40%的开发时间；一些可靠性要求非常高

的软件,甚至会占到60%);另外,测试中的许多操作是重复的、非智力性的和非创造性的,但要求准确细致。最后,那些固定的、重复性的工作,可以由测试工具来完成,这样就使得测试人员能有更多的时间来计划测试过程,设计测试用例,使测试更加完善。

2. 保证测试的准确性

手工测试常会犯一些人为错误,为此需要投入大量的时间和精力;而工具的特点是能保证测试的准确性,防止人为疏忽造成的错误。

测试工具可以进行部分的测试设计、测试实现、测试执行和测试比较的工作,这些工作中的很多可应用自动化技术来完成。如采用自动比较技术,可以自动完成测试用例执行结果的判断,从而避免人工比对存在的疏漏问题。设计良好的自动化测试,在某些情况下可以实现"夜间测试"和"无人测试"。在大多数情况下,软件测试自动化可以减少开支,增加有限时间内可执行的测试,在执行相同数量测试时节约测试时间。

3. 有些测试很难开展,必须使用工具(如性能测试等)

测试工具可以执行一些手工难于执行,或者是无法执行的测试。这是因为软件测试工作相当复杂,要求非常严格,很多测试在手工测试环境是无法完成的。

4. 测试工具很好地保证测试工作的规范性和一致性

软件工程最重要的内容就是管理,软件测试同样也是将管理放在第一位。

5. 测试工具体现了先进的测试思想、方法和技术

测试工具的使用能够快速地提升软件测试的专业化水平。因此,测试工具的选择和推广也越来越受到重视。

在软件测试工具平台方面,商业化的软件测试工具已经很多,如捕获/回放工具、Web测试工具、性能测试工具、测试管理工具、代码测试工具等,这些都有严格的版权限制且价格较为昂贵,但也有一些软件测试工具开发商对于某些测试工具提供了Beta测试版本以及试用版供用户有限次数地使用。另外,在开放源码社区中也出现了许多软件测试工具,已得到广泛应用并且相当成熟和完善。

2.2.3 软件测试工作的几个认识误区

随着软件规模的不断扩大,软件设计的复杂程度不断提高,软件开发中出现错误或缺陷的机会越来越多。同时,市场对软件质量重要性的认识逐渐增强。所以,软件测试在软件项目实施过程中的重要性日益突出。但是,现实情况是,与软件编程比较,软件测试的地位和作用,还没有真正受到重视,对于很多人(甚至是软件项目组的技术人员)还存在对软件测试的认识误区,这进一步影响了软件测试活动的开展和软件测试质量的真正提高。

1. 整体认识上重开发而轻测试

整个软件项目制作过程中的各种活动按重要程度进行排序,结果软件测试几乎从未名列前茅。另外,如果在软件项目面临着时间压力,必须加快研发速度时,通常软件测试会成为换取项目进度的牺牲品。重开发而轻测试在软件开发中颇为普遍。

2. 软件开发完成后进行软件测试

人们一般认为,软件项目要经过以下几个阶段:需求分析、概要设计、详细设计、软件编码、软件测试、软件发布。据此,认为软件测试只是软件编码后的一个过程。这是不了解软件测试周期的错误认识。

软件测试是一个系列过程活动,包括软件测试需求分析、测试计划设计、测试用例设计、执行测试。因此,软件测试贯穿于软件项目的整个生命过程。在软件项目的每一个阶段都要进

行不同目的和内容的测试活动，以保证各个阶段的正确性。软件测试的对象不仅仅是软件代码，还包括软件需求文档和设计文档。软件开发与软件测试应该是交互进行的，例如，单元编码需要单元测试，模块组合阶段需要集成测试。如果等到软件编码结束后才进行测试，那么，测试的时间将会很短，测试的覆盖面将很不全面，测试的效果也会大打折扣。更严重的是如果此时发现了软件需求阶段或概要设计阶段的错误，并要求修复该类错误，将会耗费大量的时间和人力。

3. 软件测试是为了证明软件的正确性

测试的目的是证伪而不是证真。事实上，要证明程序的正确性是不可能的，一个大型的集成化的软件系统不能被穷尽测试以遍历其每条路径，即使是遍历了所有的路径，错误也仍有可能隐藏。做测试的目的只是尽可能地发现错误。

4. 软件发布后如果发现质量问题，那是软件测试人员的错

这种认识严重挫伤了软件测试人员的积极性。软件中的错误可能来自软件项目中的各个环节和阶段，软件测试只能确认软件存在错误，不能保证软件没有错误，因为从根本上讲，软件测试不可能发现全部的错误。从软件开发的角度看，软件的高质量不是软件测试人员测出来的，是靠软件生命周期的各个环节和阶段设计出来的。出现软件错误，不能简单地归结为某一个人的责任，有些错误的产生可能不是技术原因，可能来自混乱的项目管理。应该分析软件项目的各个过程，从过程改进方面寻找产生错误的原因和改进的措施。

5. 软件测试要求不高，随便找个人就行

很多人认为软件测试就是安装和运行程序，点点鼠标，敲敲键盘的工作。这是由于不了解软件测试的具体技术和方法造成的。测试人员能力的高低会极大地影响测试工作的成败，一个测试者必须既明白被测软件系统的概念又要会使用工程中的那些工具，要做到这一点还需要有几年以上的编程经验。随着软件工程学的发展和软件项目管理经验的提高，软件测试已经形成了一个独立的技术学科，演变成一个具有巨大市场需求的行业。软件测试技术不断更新和完善，新工具、新流程、新测试设计方法都在不断更新，需要掌握和学习很多测试知识。所以，具有编程经验的程序员不一定是一名优秀的测试工程师。软件测试包括测试技术和管理两个方面，完全掌握这两个方面的内容，需要很多测试实践经验和不断学习精神。

6. 软件测试是软件开发的敌人

测试人员和开发人员经常无法有效地一起工作。其实，开发和测试作为一个整体都是服务于产品，都要为产品的质量负责。因此测试应该是开发的朋友，而不是开发的敌人。

7. 软件测试是测试人员的事情，与程序员无关

开发和测试是相辅相成的过程，需要软件测试人员、程序员和系统分析师等保持密切的联系，需要更多的交流和协调，以便提高测试效率。另外，对于单元测试主要应该由程序员完成，必要时测试人员可以帮助设计测试用例。对于测试中发现的软件错误，很多需要程序员通过修改编码才能修复。程序员可以通过有目的地分析软件错误的类型、数量，找出产生错误的位置和原因，以便在今后的编程中避免同样的错误，积累编程经验，提高编程能力。

8. 项目进度吃紧时少做些测试，时间富裕时多做测试

这是不重视软件测试的表现，也是软件项目过程管理混乱的表现，必然会降低软件测试的质量。一个软件项目的顺利实现需要有合理的项目进度计划，其中包括合理的测试计划，对项目实施过程中的任何问题，都要有风险分析和相应的对策，不要因为开发进度的延期而简单地缩短测试时间，减少人力和资源。因为缩短测试时间带来的测试不完整，对项目质量的下降引起的潜在风险，往往造成更大的浪费。克服这种现象的最好办法是加强软件过程的计划和控

制，包括软件测试计划、测试设计、测试执行、测试度量和测试控制。

9. 软件测试是没有前途的工作，只有程序员才是软件高手

由于我国软件整体开发能力比较低，软件过程很不规范，很多软件项目的开发都还停留在“作坊式”和“垒鸡窝”阶段。项目的成功往往靠个别全能程序员决定，他们负责总体设计和程序详细设计，认为软件开发就是编写代码，给人的印象往往是程序员是真正的“牛人”，具有很高的地位和待遇。因此，在这种环境下，软件测试很不受重视，软件测试人员的地位和待遇自然就很低了，甚至软件测试变得可有可无。随着市场对软件质量要求的不断提高，软件测试将变得越来越重要，相应的软件测试人员的地位和待遇将会逐渐提高。在微软等软件开发过程比较规范的大公司，软件测试人员的数量和待遇与程序员没有多大差别，优秀测试人员的待遇甚至比程序员还要高。软件测试将会成为一个具有很大发展前景的行业，软件测试大有前途，市场需要更多具有丰富测试技术和管理经验的测试人员，他们同样是软件专家。

10. 软件测试就是程序测试，测试发现了错误就说明是程序员所编写的程序有问题

在软件项目开发的整个过程中，前一阶段工作中发生的问题如未及时解决，很自然地要影响到下一个阶段。表现在程序中的缺陷可能是设计阶段甚至于需求分析阶段的问题所致，大多数软件缺陷并非来自编码过程，从小项目到大项目基本上都如此。即使是针对源程序进行测试所发现的故障，其根源也可能存在于软件开发前期的各个阶段。因此不能简单地把程序中的错误全都归罪于程序员。

11. 期望用测试自动化代替大部分人工劳动

目前，很多的企业首先是从节约成本的角度考虑去引入测试自动化工具的。自动化测试工具的确能用于完成部分重复、枯燥的手工作业，但不要指望它来代替人工测试。一般来讲，产品化的软件更适于功能测试的自动化，由标准模块组装的系统更好，因为其功能稳定，界面变化不大。性能测试似乎更加依赖于自动化测试工具（如模拟多个虚拟用户和收集性能指标等），但考虑购买时也要注意，假设一个测试工具不支持.Net，也许它就不一定适用于贵公司。

不要因为自动化测试工具前面有“自动化”三个字就认为它的主要目的是来代替手工劳动的。不要因为大部分测试自动化工具有“功能”和“性能”两种脚本类型，就认为自动化测试不是做用户界面验证就是做产品性能检测的。不需要昂贵的自动化工具，某些重复测试任务也可以考虑用编程方式实现自动化。如：20％的测试用例自动化执行，用于覆盖80％的用户操作密集的功能和核心商业逻辑（如工资计算准确度要求高，虽然每月才执行一次）。实现功能测试自动化来完成重复、枯燥的回归测试任务，引入性能测试自动化工具来改善测试的广度和深度。另外，由于机器和脚本是客观的，它总是会完成我们所分配的所有任务，而没有半点遗漏，这有助于我们真正掌握和控制我们的回归测试覆盖率。

12. 所有软件缺陷都可以修复

在软件测试中，有一种令人沮丧的现实是，即使拼尽全力，也不是所有的软件缺陷都能修复。但是，这并不意味着软件测试未达到目的，或者项目小组将发布质量欠佳的产品。事实上，也不需要对所有的软件缺陷进行修复。

13. 认为软件测试文档不重要

对于软件测试，4个最典型的书面文档是测试计划、测试用例、缺陷列表、测试报告。

测试计划：测试作为整个项目工程的一部分，在早期做出较为详细的测试范围、人力预算、执行时间、技术需求/培训和软硬件资源占用等方面的考虑，便于有目的、有计划地完成后面的测试工作，测试团队也能更好地与其他团队协作。后期评审中，以此为一个基线，更容易发现执行中的问题并及时作出调整。

测试用例：可以让参与测试的人员，花足够的精力，第一时间去系统地理解需求。在此基础上进行的同行评审，则可以防止需求理解得不彻底和偏差，尽早发现可能存在的测试漏洞。同时，测试用例作为新旧人员交替时知识传递的媒体，有利于新人（时常也有从其他团队借调的人员）尽快进入特定模块的测试工作，而不是被成堆的需求文档所淹没。

缺陷列表：好的缺陷管理应该引入一个软件系统，而不是使用传统意义上的 Word 或 Excel 文档。想象一下，当缺陷数目达到几百个，按照某些工业模板去生成 Bug 文件，怎能做到方便地管理、查询和分析？如果测试周期持续很长，测试人员又不止一个，报 Bug 前寻找雷同的旧记录都将非常费时、费力。

测试报告：把测试的过程和结果写成文档，并对发现的问题和缺陷进行分析，为纠正软件存在的质量问题提供依据，同时为软件验收和交付打下基础。另外，测试报告是测试阶段最后的文档产出物，一份详细的测试报告包括产品质量和测试过程的评价。测试报告基于测试中的数据采集以及对最终的测试结果分析。比如覆盖率分析、缺陷分析等。

14．期望短期通过增加软件测试投入，迅速达到零缺陷率

即使我们有充裕的资金，也不是说软件测试投入得越多越好。增加测试人力和时间上的投入，的确能帮助我们找出更多的缺陷。但两者不是一种线性关系，随着测试投入的不断放大，产品质量上升是逐渐收敛的。一个项目投入 10 个测试人员，发现了 70％的缺陷，并不表明投入 20 个人就能找出几乎所有的缺陷，也许这个数字只会是 85％。

所以，测试经理要根据公司的具体情况，如策略方针、市场定位以及产品类别等因素，来决定开发人员和测试人员的比率以及测试投入。举例来说，做一个 ERP 解决方案、一个杀毒软件乃至一个航天飞船的控制系统，对测试的要求（包括技术结构和资源占用等）肯定是不一样的。

另外，软件质量的好坏，和整个软件开发过程的优劣是密不可分的，决不是仅仅通过加强测试就可以完全控制的。

15．规范化软件测试是增加项目成本

增加软件测试人员和预留项目测试时间，表面上看是增加了人员成本或延长了项目周期，为此会投入更多的项目资金。然而，越早发现软件中存在的问题，开发费用就越低。美国质量保证研究所对软件测试的研究结果表明：在编码后修改软件缺陷的成本是编码前的 10 倍，在产品交付后修改软件缺陷的成本是交付前的 10 倍；软件质量越高，软件发布后的维护费用越低。

所以，即使我们的客户（包含本公司市场部人员）有最好的忍耐力，或愿意免费充当我们的测试队伍，前期找出缺陷也是会省去很多的后期排错和维护的费用。加上由于产品不成熟而造成市场声誉受损，单从经济效益方面来考虑，前期足够的测试也是值得投入的。

2.3　软件测试职业

由于近年来我国软件行业的产业升级、软件开发模式的升级（软件开发的单打独斗升级为工业化、流水线式的生产模式），软件测试逐步成为软件开发企业必不可少的质量监控环节，贯穿于软件产品研发周期内每一个环节和阶段，并在整个软件开发中占据相当大的比重。在软件产业发达国家，软件企业一般是把 40％的工作花在测试上，测试人员和开发人员之比平均在 1∶1 以上，软件测试费用占整体开发费用的 30％～50％，对于高可靠性、高安全性的软件，测试费用则相当于整个软件项目开发费用的 3～5 倍，这些充分说明了软件测试的重要性。目前我国无论是政府、企业还是高等院校，对软件测试工作和人才培养一直不够重视，大家重开

发、轻测试，以致我国在软件测试的投入远远低于软件开发上的投入，远远低于软件产业发达国家在软件测试上的投入。软件测试人才的培养数量较发达国家以及我国产业升级相当滞后，这就形成了软件测试人才的供给远小于需求的现状。

最近几年我国的媒体时常报道：中国软件测试人才缺口 20 万，软件测试工程师将成为未来 10 年最紧缺的人才之一，众多国内外优秀企业对高端测试人才年薪 10 万、15 万、20 万元的招聘需求比比皆是。由此可见，软件测试目前是一个朝阳行业。但是，近几年在北京、上海、深圳举办的各种大型招聘会上，多家企业纷纷打出各类高薪招聘软件测试人员的海报，出人意料的是，收到的简历尚不足招聘岗位数的 50%，而合格的竟不足 30%。这说明了我们的软件测试从业人员还有很大一部分不满足当今社会的需求，高素质、专业化的软件测试人才非常紧缺，具有一定测试经验的软件测试工程师很受市场青睐，供不应求。而另一层含义是，我们还有很大的提升空间。

目前，软件测试工作越来越受重视，现在测试人员的待遇和开发人员的待遇逐渐接近。

2.3.1 软件测试职业发展

软件测试职业发展方向，与其他职业发展是相吻合的，也可以分为管理路线、技术路线、管理+技术路线。

软件测试人员的职业发展是基于软件测试团队这个组织的人员角色转变或升迁的。也就是说，软件测试人员要确立在测试团队这个组织中的职业规划图（即个人 Roadmap），然后基于这个 Roadmap 再确立各个角色的职责以及各角色之间的相互联系和发展顺序。这样，软件测试人员在测试团队中的发展目标也就确立了。

1. 测试团队的基本构成

作为一个测试团队或一个比较健全的测试团队应该具有下面这些人员角色。

（1）测试经理：主要负责人员的招聘、培训和管理，以及资源调配、测试方法改进等。

（2）实验室管理人员：设置、配置和维护实验室的测试环境，如服务器和网络环境等。

（3）测试配置管理人员：审查流程，并提出改进流程的建议；建立测试文档所需的各种模板，进行测试的配置管理，检查软件缺陷描述及其他测试报告的质量等。

（4）测试组长：业务专家，负责项目的管理，包括测试计划的制订、项目文档的审查、测试用例的设计和审查、测试任务的安排、与项目经理及开发组长的沟通等。

（5）一般（初级）测试工程师：编写、执行测试用例，编制有关的测试文档或开展其他相关的测试任务。

对于比较大规模的测试团队，测试工程师分为三个层次：初级测试工程师、测试工程师、资深（高级）测试工程师等，同时还设立自动化测试工程师、系统测试工程师和架构工程师。

对于规模很小的测试小组，可能没有设置测试经理，只有测试组长。这时测试组长承担测试经理的部分责任，如参加面试工作、资源管理、团队发展等，并且要做内审员的工作，检查软件缺陷描述及其他测试报告的质量等。资深测试工程师不仅要负责设计规格说明书的审查、测试用例的设计等，还要设置测试环境，即承担实验室管理人员的责任。对于一些大型的企业或专业的测试团队，除了测试经理外，在他之上还有测试总监。测试总监是在公司或企业层面上对整个测试工作进行管理，包括行政上和技术上的管理。

2. 测试人员职位及责任

基于前面测试人员角色的划分，对应其职位，从一般（初级）测试工程师开始，再到资深测试工程师，最后到测试经理，自底向上地了解他们的责任。测试工程师虽然和初级测试工程师

责任不一样，但测试工程师肯定能做好所有要求初级测试工程师做好的工作。

不同层次的测试工程师责任有一定的区别，但都是技术工作，主要任务是设计和执行各种测试任务，是测试工作的基础。

1）初级测试工程师

初级测试工程师的责任比较简单，还不具备完全独立的工作能力，需要测试工程师或资深测试工程师的指导，要求比较低。初级测试工程师的主要责任是：①了解和熟悉产品的功能、特性等；②验证产品在功能、界面上是否和产品规格说明书一致；③按照要求，执行测试用例，进行功能测试、验收测试等，并能发现所暴露的问题；④清楚地描述所出现的软件问题；⑤努力学习新技术和软件工程方法，不断提高自己的专业水平；⑥使用简单的测试工具；⑦接受测试工程师的指导，执行主管所交代的其他工作。

2）测试工程师

测试工程师的适用范围是有软件测试近3年经历的常规测试从业者。测试工程师的责任相对多些，具有独立的工作能力，其主要工作内容是按照测试主管或测试组长（即直接上司）分配的任务计划，熟悉测试流程、测试方法和技术，编写测试用例、执行测试用例（包括参与自动化测试）、提交软件缺陷，包括提交阶段性测试报告、参与阶段性评审等。

测试工程师以执行测试为主，其主要责任是：

① 熟悉产品的功能、特性，审查产品规格说明书。

② 根据需求文档或设计文档，可以设计功能方面的测试用例。

③ 根据测试用例，执行各种测试，发现所暴露的问题。

④ 全面使用测试工具，包括测试脚本的编写。

⑤ 安装、设置简单的系统测试环境。

⑥ 报告所发现的软件缺陷，审查软件缺陷，跟踪缺陷修改的情况，直到缺陷关闭。

⑦ 撰写测试报告。

⑧ 负责对初级测试工程师进行指导，执行主管所交代的其他工作。

3）资深测试工程师

资深测试工程师的适用范围是具有3～5年职业经验的测试从业者。资深测试工程师不仅具有良好的技术、产品分析能力、解决问题能力、丰富的测试工作经验，而且有较好的编程、自动化测试经验，熟悉测试流程、测试方法和技术，解决测试经理工作中可能遇到的各种技术问题。

资深测试工程师的工作内容是根据项目经理或测试经理的计划安排，调配测试工程师执行模块级或项目级测试工作，并控制与监督软件缺陷的追踪，保证每个测试环节与阶段的顺利进行。严格来说，这个级别更多属于测试的设计者，因为企业的测试流程搭建是由更高级别的测试经理或相关管理者来做的，资深测试工程师负责该流程的具体实施；而更多的工作，是思考如何对软件进行更加深入、全面的测试。资深测试工程师比较有创造性的工作内容就是测试设计，而恰恰很多公司忽略了或没有精力来执行此工作内容。应该说，在一个企业里做了3年左右测试工作的人员，很容易晋升到该职位，而之所以晋升，是与个人测试技术的过硬、测试方法的丰富，加上对测试流程的监控力与执行力的职业素质息息相关。

资深测试工程师的主要责任是：

① 负责系统中一个或多个模块的测试工作。

② 制定某个模块或某个阶段的测试计划、测试策略。

③ 设计测试环境所需的系统或网络结构，安装、设置复杂的系统测试环境。

④ 熟悉产品的功能、特性,审查产品规格说明书,并提出改进要求。
⑤ 审查代码。
⑥ 验证产品是否满足规格说明书所描述的需求。
⑦ 根据需求文档或设计文档,设计复杂的测试用例。
⑧ 负责对测试工程师的指导,执行主管交代的其他工作。

4)测试实验室管理员

测试实验室管理员主要负责建立、设置和维护测试环境,保证测试环境的稳定运行,其主要责任是:
① 负责测试环境所需的网络规划和建设,维护网络的正常运行。
② 建立、设置和维护测试环境所需的应用服务器和软件平台。
③ 申请所需的硬件资源、软件资源,协助有关部门进行采购、验收。
④ 对使用实验室的硬件、软件资源的权限进行设计、设置,保证其安全性。
⑤ 安装新的测试平台、被测系统等。
⑥ 优化测试环境,提高测试环境中网络、服务器和其他设备运行的性能。

5)测试配置管理人员

测试配置管理人员在QA工作中起着很重要的角色,负责测试产品的上载、打包和发布等测试配置管理工作,其主要责任是:
① 负责源程序代码管理系统的建立、管理和维护。
② 文件名定义规范,建立合理的程序文件结构和存储目录结构。
③ 为程序的编译、连接等软件包的构造建立自动处理文件。
④ 保证测试最新的产品包上载到相应的服务器上,并确认各模块或组件之间相互匹配。
⑤ 每天为不同项目新的或修改的代码重新构造新的软件包。
⑥ 确保不含病毒,不缺图片和各种文件。
⑦ 文件包的接收、发送、存储和备份等。

6)测试组长

测试组长一般要有3年以上的测试经验,1年以上的2~7人测试项目管理经验。具备资深测试工程师的能力和经验,熟练掌握全面的测试理论,能够很好地使用相关测试技术和工具,熟悉软件测试流程,具有优秀的测试文档编写能力,包括测试计划、测试方案、测试用例、测试报告等。测试组长可能在技术上相对弱些,不是小组内最强的,但他能够带领团队内的测试工程师,执行所负责软件的测试计划,跟踪并报告测试计划的执行进度。另外,测试组长要有较强的交流和沟通能力。测试组长的主要责任是:
① 负责测试项目的全面管理,包括测试小组的业务管理和人员管理。
② 与客户沟通交流,建立良好的合作氛围。
③ 制定测试计划,跟踪并报告测试计划的执行进度。
④ 执行测试并及时反馈项目状态,分析项目风险。
⑤ 负责或者组织本项目组的培训工作。
⑥ 发展团队核心竞争力并扩展团队工作内容。

7)测试经理

测试经理是更高级别的测试管理者。对于大中型软件公司,该职位尤为重要,并且对其职业要求也比较高,一般适合4~8年的测试从业者,在管理与技术能力均成熟的情况下,可以结合具体环境晋升到该级别。

测试经理的主要工作在团队、资源和项目等的管理上，与测试组长有较大的不同。测试组长主要集中在项目管理上，一般不负责测试人员的招聘、流程定义等管理工作，而是偏重技术。测试经理对产品的质量负全面责任，有责任向公司最高管理层反映软件开发过程中的管理问题或产品中的质量问题，使公司能全面掌握生产和质量状况。

测试经理的主要职责是：

① 负责企业级或大型项目级总体测试工作的策划与实施。测试经理除了需要统筹整个企业级或项目级测试流程外，还要对不同软件架构、不同开发技术下的测试方法进行研究与探索，为企业的测试团队成员提供指导与解决思路，同时还要合理调配不同专项测试的人力资源（如业务测试工程师、自动化测试工程师、"白盒"测试工程师、性能测试工程师等），对软件进行全面的测试。

② 负责被测项目（测试/质量/开发）内的整个开发生命周期业务，包括项目成本分析、进度安排、计划和人员分工。

③ 负责与客户（甚至与开发团队）的交流与沟通。

④ 负责部分的销售性或技术支持性工作。

8）测试总监

测试总监属于常规发展路线的最高域，该职位一般在大型或跨国型软件企业，或者专向于测试服务型企业设立，一般设立测试总监的企业，该职位都相当于技术总监或副总的级别，是企业级或集团级测试工作的最高领导者，驾驭着企业全部的测试和与测试相关资源，管理着企业的全部测试及质量类工作。而其职业要求，也是技术与管理双结合。

测试总监的主要职责是：

① 根据企业年度预算目标，制定本部门年度人员配置计划和费用预算。

② 负责项目测试计划的审核，并带领测试团队设计、执行、优化测试过程，丰富测试手段，引入新的测试框架和测试策略，持续改进软件测试过程，确保项目测试符合流程与规范。

③ 与其他测试人员、开发人员、项目管理人员沟通和协作，推动整个项目的顺利进行，同时组织协调资源，确保本部门各项测试工作按计划完成。

④ 制定本部门的《年度绩效目标责任书》，并跟进绩效目标完成。

⑤ 负责本部门测试团队的建设和管理，包括测试团队中的各类测试人员的培训、指导、培养等管理工作，不断提升公司的测试能力，带领和指导整个测试团队科学、规范、合理、高效地协同工作。

⑥ 维护测试流程，统计和分析测试结果，提高及改进测试效率和质量。

⑦ 负责本部门测试工具、测试环境及测试实验室的规划、建设和使用，包括测试技术的研究与测试工具的研究。

⑧ 负责公司各项规章制度在本部门的监督执行。

2.3.2　软件测试人员应具备的素质

在软件开发和软件测试中，软件开发人员和测试人员的素质（包括心理素质）对软件质量的影响都是很大的，如：开发人员认为他不会犯错，但任何人都可能犯错；开发人员认为这种错误不能算作错误，事实上质量是由用户来评价的；开发人员认为发现他的错误是对他工作的否定，实际上这是对他工作的一个很好帮助。

测试人员分为两类：测试工具软件开发工程师和软件测试工程师。前者介于软件开发工程师和软件测试工程师之间，负责写测试工具代码，并利用测试工具对软件进行测试，或者开

发测试工具为软件测试工程师服务；后者负责理解产品的功能要求，然后对其进行测试，检查软件有没有错误，决定软件是否具有稳定性，并写出相应的测试计划和测试用例。为了方便，我们将软件测试工程师称为软件测试员。

前面我们提到，软件测试是一项复杂而艰巨的任务，软件测试员的目标是尽早发现软件缺陷，以便降低修复成本。软件测试员是客户的眼睛，是最早看到并使用软件的人，所以应当站在客户的角度，代表客户说话，及时发现问题，力求使软件功能趋于完善。

很多比较成熟的软件公司都把软件测试视为高级技术职位。软件测试员的工作与开发人员的工作对软件开发所起的作用是相当的。虽然软件测试员不一定是一个优秀的开发人员，但是作为一个出色的软件测试员应当具备丰富的编程知识，掌握软件编程的基础内容，了解软件编程的过程，这无疑对出色完成软件测试任务具有很大的帮助。

对于软件测试员，技术能力是相当重要的，但他们的素质培养更为重要，而这些素质具体地体现在以下几个方面。

1. 对软件测试工作有正确的认识

软件测试员可能认为软件测试不可能发现所有错误而责任心不够，这需要对他们进行职业教育，实施激励措施；另外，很多软件测试员认为测试没有创造性、枯燥，这时就需要引导他们不断总结测试经验，培养发现问题的敏锐度，提升个人价值和权威；还有，软件测试员可能认为测试所需要或用到的技术比开发人员差，自信心不足，这就需要消除他们认识上的错误。测试是技术和经验的结合，软件测试员能够多快好省地发现错误或缺陷，就能够最大限度地节省时间和费用，从而创造巨大的价值；最后，软件测试员认为软件测试就是给人挑毛病，招人厌恶，这就需要对软件测试员进行职业教育，让他们意识到软件测试员的任务就是站在使用者的角度上，代表用户通过不断地使用和攻击刚开发出来的软件产品尽量多地找出产品存在的问题或错误(Bug)，用户满意就是他的成功。

另外，在测试过程中要做到对事不对人。也就是说，软件测试员应该把精力集中在查找错误上面，而不是放在找出是开发小组中哪个成员引入的错误。这样可以保证测试的否定性结果只是针对产品，而不是针对编程人员，也就是说要使用一种公正和公平的方式指出具体错误，这对于测试工作是有益的。一般来说，武断地对产品进行攻击是错误的。

最后，测试工作很容易使人变得懒散。只有那些具有自我督促能力的人才能够使自己的工作高质量完成。

2. 具有较强的沟通能力、外交能力和移情能力

优秀的软件测试员必须能够同测试涉及的所有人进行沟通，具有与技术和非技术人员的交流能力。机智老练和外交手法有助于维护与开发人员的协作关系，幽默感同样也是很有帮助的。测试工程师既要可以和用户谈得来，又要能同开发人员说得上话，要达到这个目标是很难的，因为这两类人没有共同语言。和用户谈话的重点必须放在系统可以正确地处理什么和不可以处理什么上，尽量不使用专业术语。而和开发者交流时，要尽量使用专业术语，对用户反馈的相同信息，软件测试员必须重新组织，以另一种方式表达出来。测试小组的成员必须能够同等地同用户和开发者沟通。特别是与开发人员沟通时，软件测试员要善于表达观点，表明软件缺陷为何必须修复，并通过实际演示力陈观点。

当软件测试员告诉某人出了错时，最好使用一些外交方法。如果采取的方法过于强硬，对测试者来说，在以后和开发部门的合作方面就相当于“赢了战争却输了战役”。在遇到狡辩的情况下，一个幽默的批评将是很有帮助的。

和被测软件系统开发有关的所有人员都处在一种既关心又担心的状态之中。用户担心将

来使用一个不符合自己要求的系统，开发者则担心由于系统要求不正确而使他不得不重新开发整个系统，管理部门则担心这个系统突然崩溃而使它的声誉受损。软件测试员必须和每一类人打交道，因此需要测试小组的成员对他们每个人都具有足够的理解和同情，具备了这种能力可以将软件测试员与相关人员之间的冲突和对抗减少到最低程度。

3. 掌握比较全面的技术

很多情况下，开发人员对那些不懂技术的人持一种轻视的态度。一旦测试小组的某个成员做出了一个比较明显的错误断定，很可能会被夸张地到处传扬，那么测试小组的可信度就会受到影响，其他正确的测试结果也会受到质疑。再者，由于软件错误通常依赖于技术，或者至少受构造系统所使用的技术的影响，所以软件测试员需要掌握编程语言、系统构架、操作系统的特性、网络、数据库的特性功能和操作等知识，了解系统是怎样构成的，清楚被测软件系统的概念和用到的技术，能够建立测试环境、编写测试脚本，还会使用软件工程工具。要做到这些，需要有几年以上的编程经验和对技术及应用领域的深刻理解。

4. 测试中要做到“五心”

专心：主要指软件测试员在执行测试任务的时候要专心，不可一心二用。经验表明，精力高度集中不但能够提高效率，还能发现更多的软件缺陷，业绩最棒的往往是团队中做事精力最集中的那些成员。

细心：主要指执行测试工作时要细心，认真执行测试，不可以忽略一些细节。某些缺陷如果不细心很难发现，例如一些界面的样式、文字等。

耐心：很多测试工作有时候显得非常枯燥，需要很大的耐心才可以做好。如果比较浮躁，就不会做到“专心”和“细心”，这将让很多软件缺陷从你眼前逃过。

责任心：责任心是做好工作必备的素质之一，软件测试员更应该将其发扬光大。如果测试中没有尽到责任，甚至敷衍了事，这将会把测试工作交给用户来完成，很可能引起非常严重的后果。

自信心：自信心是现在多数软件测试员都缺少的一项素质，尤其在面对需要编写测试代码等工作的时候，往往认为自己做不到。要想获得更好的职业发展，软件测试员们应该努力学习，建立能“解决一切测试问题”的信心。

5. 要有很强的记忆力、怀疑精神和洞察力

好的测试者应有能力将以前遇到过的类似错误从记忆深处挖掘出来，这一能力在测试过程中的价值是无法衡量的。因为许多新出现的问题和我们已经发现的问题相差无几。

通常，开发者会尽他们最大努力将所有的错误解释过去。软件测试员必须听每个人的说明，但必须保持高度警惕、怀疑一切，直到自己的分析结果或亲自测试之后，才能做出决定。

一个好的软件测试员具有“测试是破坏”的观点，捕获用户观点的能力，追求质量、关注细节的能力，以及应用的高风险区的判断能力，这样就有可能进行针对性的测试。

6. 具有探索、创新和挑战精神，努力追求完美

首先，软件测试员不要害怕进入陌生环境，要勇于探索。当然，前提是软件测试员要有较强的学习能力，可以用最快的速度进入一个新的行业领域。

其次，软件测试员要能够想得出富有创意甚至超常的手段来寻找软件中潜在的各种错误和缺陷。

最后，优秀的软件测试员在开发测试用例时要有敢于挑战的精神，要想方设法找到隐藏在深处的错误和覆盖所有的分支及路径。

通常在测试的过程中，软件测试员常常会碰到转瞬即逝或者难以重建的软件缺陷，这时候

不要心存侥幸，而是要尽一切可能去寻找，尽力接近目标，力求完美。

7. 软件测试员在测试时要注意的事项

软件测试员在测试时要注意以下事项。

(1) 永远不要许诺或保证什么。在任何时候都不要表露出有了软件测试员或者有了像你一样的软件测试员，产品绝对没有任何问题了。这是自己打自己的嘴，软件测试员要给自己留退路，要表露出谦虚的一面，“尽量少在用户使用时发现问题”，“我会竭尽全力做好测试工作”。

(2) 文档反映了自己的精神面貌。软件测试员的任何文档代表的是他本人，所以文档一定要写得漂亮，即要求格式、版面整齐，没有错别字，语言通顺，表达清楚，没有歧义，一般的技术人员都能读懂你的文档。

(3) 要学会逆向思维。开发人员一般都是从正面满足需求，比较少去考虑不满足需求的部分，软件测试员就要逆向思维考虑，有哪些是开发人员没有考虑到的、不满足需求的部分。

(4) 编写缺陷一定要保证重现。在保证重现缺陷的时候，要注意缺陷不要描述太啰嗦，一般在 3 个步骤要完成操作。

(5) 测试要依据需求，关注对用户不利的缺陷。离开了需求，就无法对被测项目进行测试。另外，在测试中要更多地考虑用户能否正确、完整地使用被测软件，用户使用这套软件能够给他们的工作带来什么样的好处，不要过多考虑用户不在意的问题。

(6) 尽量使用测试工具。完全的手工测试过程是非常浪费时间和资源的，所以软件测试员应该根据公司的实际情况适当地引入测试工具。一般情况首先引入的是测试管理工具，把整个测试过程管理起来，然后考虑其他测试工具。

(7) 牢记服务意识。软件测试员是服务人员，整个项目组的人都是软件测试员服务的对象，针对不同的人，我们应该提供不同的帮助与协助。

总之，测试工作对软件测试员有很高的素质要求，如：包括坚持原则在内的责任心，包括好奇心在内的怀疑精神和学习能力，包括与用户和项目组人员在内的沟通能力，包括耐心和记忆力在内的专注力，包括经验、逻辑思维能力和敏感度在内的洞察力，以及团队精神等。另外，除了素质要求外，对软件测试员还有很高的技术要求，如计算机操作能力、测试环境搭建能力、一定的编程基础(如对编程机制、实现架构有一定的了解，会对测试工作很有帮助，发现很多更深层次的问题)、测试基本理论和方法(如掌握测试的基本流程与基本概念，较强的文档能力，会撰写测试报告，会设计、编写测试用例，熟悉测试工具，能够执行测试并跟踪错误等)。

2.3.3 软件测试的就业前景

中商情报网数据显示：2011 年中国软件产业企业个数有 22 788 家，其中，软件业务收入为 18 467.93 亿元，同比增长率为 32.40%；软件产品收入为 6157.78 亿元，同比增长率为 28.50%；信息系统集成服务收入为 3921.36 亿元，同比增长率为 28.40%。

据前程无忧招聘网统计，目前，国内 120 万软件从业人员中，真正能担当软件测试职位的不超过 5 万人，软件测试人才缺口已超过 20 万并向 30 万大关急速挺进。在中华英才网发布的 2010 年十大热门职业中，软件测试工程师也位居三甲之列。人才的极度匮乏令许多 IT 企业不得不延缓甚至停止项目，为企业发展带来消极影响，但对人才就业却有积极意义，人才供不应求让软件测试人员的就业竞争压力明显小于同类其他职业。

微软公司软件测试工程师对外透露，在微软内部，软件测试工程师和开发工程师的比例基本维持在 1∶1，而国内其他软件企业中这一比例却仅在 1∶5 至 1∶8 之间。“招个高水平的软件测试人员比招博士还难!”不少企业发出类似的感叹。但随着中国改革开放的不断深入，

不难看出，汽车电子、家用电子、医疗电子等产品都有了飞速的发展，软件测试行业更是如此。相信不久的将来，国内软件测试人员与开发人员的比例将会达到甚至超出 1∶1。

为了吸引更多的人才，企业纷纷采取高薪策略。据统计，刚入行的软件测试人员，起步月薪大多在 3000～5000 元，高于同龄人 1000～2000 元的薪资水平，另外还可享受带薪年假、内部培训、住房公积金等福利待遇，工作 2～3 年月薪大约在 8000～13 000 元，甚至超出很多相同服务年限的软件开发人员的薪资水平。

与企业高薪难觅良将形成鲜明对比的是，高学历人才难找合适的工作，薪酬持续走低。据中华英才网的调查显示，我国毕业生月平均收入涨幅缓慢。受市场规律影响，人才势必会朝需求旺盛的职业流动。但这流动的过程并非一帆风顺，人才首先面对的就是职业专业性的考验，这一点在软件测试人才身上体现得尤为明显。

软件测试人员最大的优势就是发展方向多，一方面，由于工作的特殊性，软件测试人才更强调经验积累，测试人员不但需要对软件的质量进行检测，而且对于软件项目的立项、管理、售前、售后等领域都要涉及，这样在几年的测试经验背景下，可以逐步转向管理或者资深测试工程师，担当测试经理或者 QA 部门主管，也可以横向发展为项目经理，所以发展前景广阔，职业寿命更长；另一方面，由于国内软件测试工程师人才奇缺，并且一般只有大中型企业才会单独设立软件测试部门，所以很有保障，待遇普遍较高。正因为如此，软件测试已经成为现在求职者关注的职业。

目前，软件测试行业正处在一个蓬勃发展的初期，未来 5～10 年将是一个发展的高潮期。

习题

1. 什么是软件测试？软件测试包含哪些概念？如何对软件测试的质量进行度量？软件测试与软件开发之间有什么关系？

2. 软件测试工作包括哪些内容？它是怎样的一个流程？测试工具对测试工作有何支持？目前人们对测试工作有哪些不正确的认识？

3. 你怎么看待软件测试？软件测试是一个什么样的行业？如果你想从事软件测试工作，你怎样做职业准备或规划？

4. 通常企业对软件测试工程师的素质和技能要求有哪些？怎样才能在软件测试职业上获得最佳发展？

5. 列出国内外有关软件测试的网站(用搜索引擎)，并描述它们各自的特点。

6. 给出国内市场对软件测试工程师的需求情况以及能力的要求。

7. 软件测试是一个独立的过程，与开发人员无关，这种说法正确与否，为什么？

第3章 生命周期软件测试方法

如同软件生命周期一样，我们也可以将软件测试阶段按照软件生命周期去划分，形成了基于生命周期的软件测试(后面简称生命周期测试)。此时，软件测试可以划分为测试需求分析、测试计划、测试设计、测试开发、测试执行、测试评估。这样，生命周期测试方法将测试延伸到需求分析、设计审查活动中去，也就是将“质量保证”的部分活动归为测试活动，真正体现了“尽早地和不断地进行软件测试”的原则，确保了对软件生命周期的每个阶段进行质量管理，并通过测试手段实现对各个阶段的质量保证。

3.1 生命周期测试的概念

按照传统的软件生命周期的观点，测试是在编程活动之后进行的，是软件开发的最后一个阶段。随着人们对软件工程化的重视以及软件规模的日益扩大，软件分析、设计的作用越来越突出，而且有资料表明，60%以上的软件错误并不是程序错误，而是需求分析和系统设计错误。如，从 IBM 提供的数据来看，对一个大约 60 个缺陷/千行的软件，2/3 的缺陷产生在需求和设计阶段，而在需求和设计阶段发现缺陷并进行修正的花费最小，否则，到了系统测试阶段来修正所发现的缺陷，花费是以上的 10 倍，到了产品发布阶段来修正所发现的缺陷，花费将是 100 倍。这说明，在需求和设计阶段就能发现软件的缺陷，那么修正所需的花费比在编程完成后再进行测试所需的花费少很多。因此，做好软件需求和设计阶段的测试工作就显得非常重要，这就使得传统的测试概念扩大化，从而提出了软件生命周期测试的概念。

生命周期测试伴随着整个软件开发周期，此时测试的对象不仅仅是程序，需求、功能和设计同样要测试。如在项目需求分析阶段就要开始参与，审查需求分析文档、产品规格说明书；在设计阶段，要审查系统设计文档、程序设计流程图、数据流图等；在代码编写阶段，需要审查代码，看是否遵守代码的变量定义规则、是否有足够的注释行等。测试与开发同步进行，有利于尽早地发现问题，同时缩短项目的开发建设周期。

3.1.1 生命周期测试的工作划分

生命周期测试意味着测试与软件开发平行，在软件开发的所有阶段进行测试，确保在尽可能早的阶段点去修正缺陷，用来减少测试成本。与软件开发一样，生命周期测试需要正式的测试流程来支持。即在软件开发团队组建时，测试小组也同时建立，在一个项目开始时，测试计划和测试条件也随着开始，并在生命周期的各阶段结束点测试系统，以确保能正确地开发系统，和尽可能在生命周期的最早的可能点发现软件的缺陷。(图 3-1)

图 3-1 表示当项目开始时，系统开发过程和系统测试过程同时开始，开发小组开始系统开发过程，而系统测试小组开始计划系统测试过程。两个小组在同一点开始，使用相同的信息。

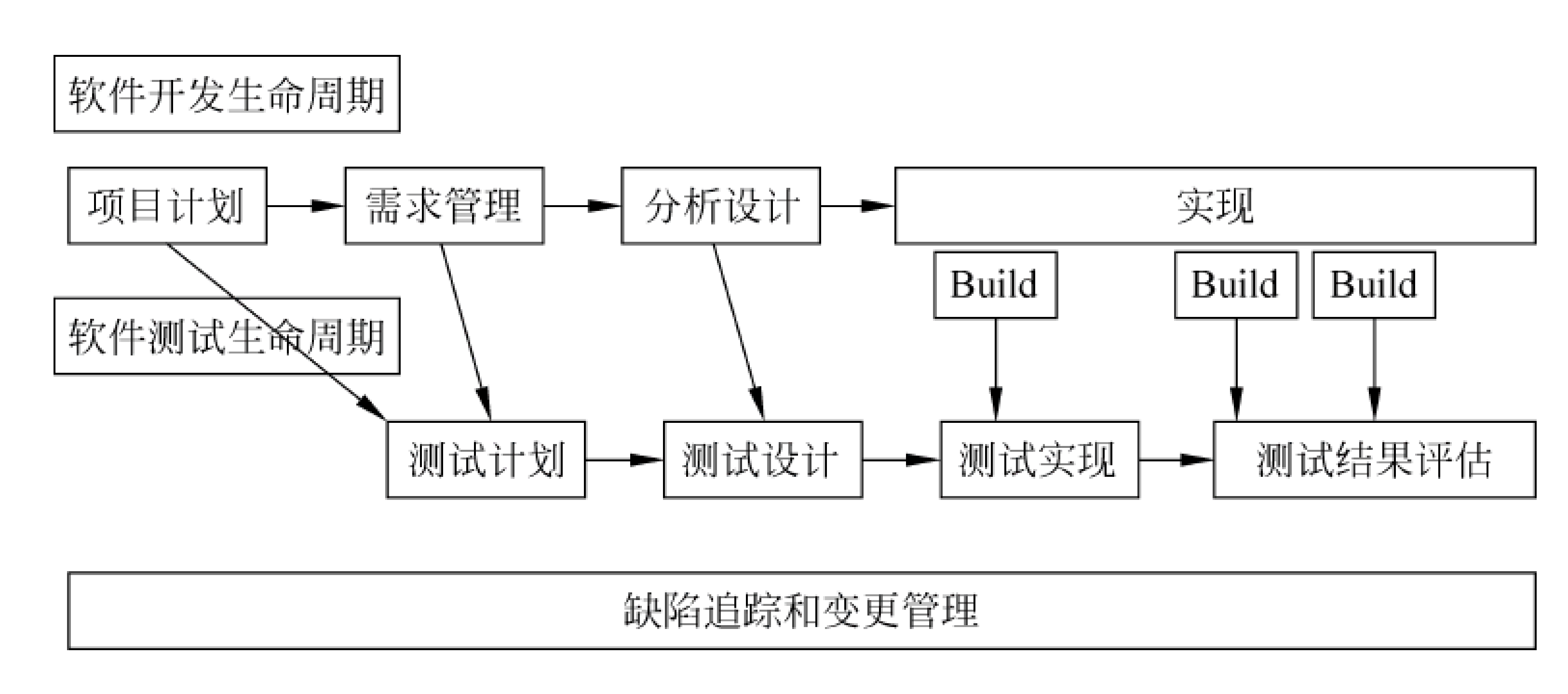


图 3-1　生命周期测试概念

测试小组在开发过程的若干个预定点对系统进行连续的测试，检查开发过程的结果。软件生命周期中要进行的测试如表 3-1 所示。

表 3-1　生命周期各阶段测试工作划分

生命周期阶段	验 证 活 动
需求	决定验证的方法 决定需求的充分程度 生成功能测试 决定与需求符合的设计
设计	决定设计的充分程度 生成结构和功能测试数据 决定设计与需求的一致性
编程	决定实现的充分程度 生成各种程序/单元的结构和功能测试数据 决定与设计的一致性
测试	决定测试计划的充分性 测试应用系统
安装/集成	把经测试的系统放入产品
维护	修改和重新测试

在需求阶段，重点是确认定义的需求符合机构的要求；在设计和编程阶段，重点是验证设计和程序实现了需求；而在测试和安装阶段，重点是检查实现的系统符合系统规格说明；在维护阶段，系统将重新测试以决定改变的部分和未改变的部分能继续工作。

3.1.2　生命周期测试的主要任务

基于生命周期测试方法对一个应用系统进行测试的测试工作过程是一个三维过程。一维概述测试要素，二维定义每个阶段要测试的事务，三维是一个测试计划，如图 3-2 所示。测试策略描述测试工程的总体方法和目标，描述目前在进行哪一阶段的测试(单元测试、集成测试、系统测试)以及每个阶段内在进行的测试种类(功能测试、性能测试、压力测试等)，给出为什么要执行测试和达到测试目标的最有效的途径。这通常由非常熟悉该软件的商业风险的小组开发；而测试种类/技术详细地解释测试的类型或采用的技术，说明执行什么测试和如何进行测试，由测试小组确定测试方法和技术的选择。

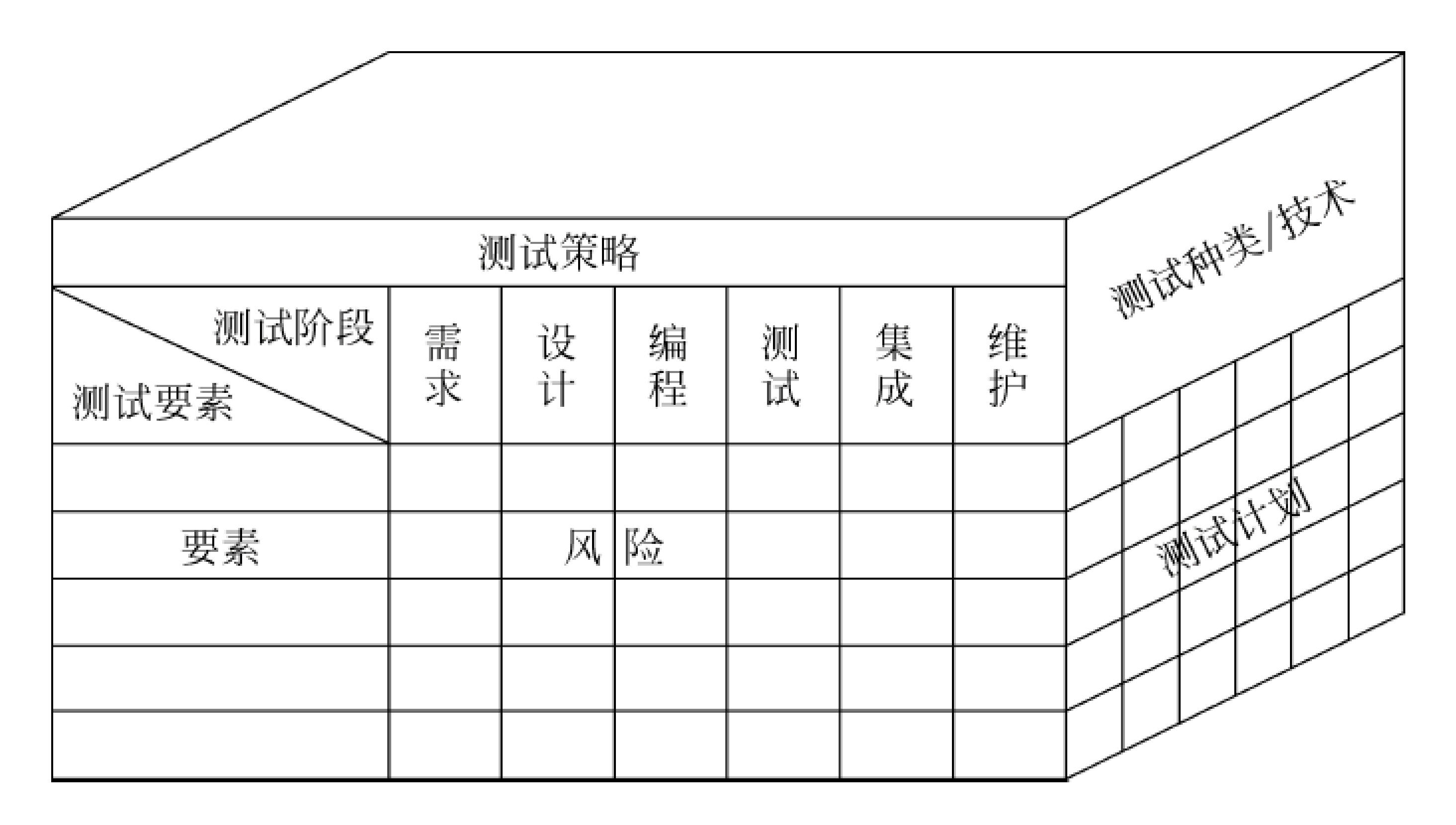


图 3-2 生命周期测试工作三维图

1. 测试要素

测试要素是计算机软件的属性，描述测试的主要目标，一个测试要素有若干个测试事件，一个事件描述测试条件和可能发生的事件，在生命周期各个阶段中，对每一个测试要素所进行的测试是不同的，也就是有不同的测试事件，它们的关系如表 3-2 所示。其中，

(1) 正确性：数据输入、过程处理和输出的正确性(IPO)。

(2) 文件完整性：文件被正确使用，恢复和存储的数据正确。

(3) 授权：特殊的授权可以执行一个特殊的操作。

(4) 进程追踪：当进程运行中，程序有能力证实进程在正常工作。

(5) 系统运行的连续性：当有非致命性问题发生后，系统有能力继续运行关键的任务。

(6) 服务水平：系统有紧急情况发生时，要求程序的输出结果不经过处理或进行简单的处理后就可以直接使用。

(7) 权限控制：防止系统被误用(意外或者有意的)。

(8) 一致性：确保最终设计和用户需求完全一致。

(9) 可靠性：在规定的时间内都可以正常运转。

(10) 易于使用：多数人均感觉易于使用。

(11) 可维护性：可以很容易地定位问题，并且进行修改。

(12) 可移植性：数据或者程序易于移植其他系统上。

(13) 耦合性：系统中的组件可以很容易地联接。

(14) 性能：系统资源的占用率、响应时间、并发处理。

(15) 操作性：易于操作(Operator)。

我们在确定测试策略时，首先选择并确定测试要素的等级(多数情况下选择 3～7 个)，并确定开发阶段；然后明确商业风险，此时开发人员、重要用户和测试人员通过评审的方式对这些风险达成一致的意见；最后把风险列表存放在需求矩阵中，矩阵中可以将风险同测试用例对应起来。

2. 风险

风险是导致失败的条件，计算机系统的风险是始终存在的。有些风险并不一定导致系统失败。我们不能消除风险，但可以减少风险发生的概率。在软件生命周期各阶段，要标识和评估计算机系统的风险，风险的概念决定测试的类型和测试工作量。决定哪些风险是可以接受的，把这些风险变成测试的领域，然后制定测试计划达到这个目标。

表 3-2 测试要素和测试事件

序号	测试要素	需求	设计	编程	测试	安装	维护
1	可靠性	建立精度等级	设计数据完整性控制	实现数据完整性控制	人工、回归和功能测试	验证安装的精度和完整性	修改精度要求
2	授权	定义授权规则	设计授权规则	实现授权规则	符合性测试	禁止改变数据	保存授权规则
3	文件完整性	定义文件完整性需求	设计文件完整性控制	实现文件完整性控制	功能测试	检查产品文件的完整性	保存文件完整性
4	审计追踪	定义重构处理需求	设计审计追踪	实现审计追踪	功能测试	记录安装审计追踪	修改审计追踪
5	处理连续性	定义失效的影响	设计中断计划	编写中断计划和过程	恢复性测试	保证以前测试的完整性	修改中断计划
6	服务级别	定义希望的服务级别	设计达到服务级别的方法	达到服务级别的服务系统	强度测试	实现故障安装计划	保存服务级别
7	存取控制	定义系统的存取	设计存取过程	实现安全过程	符合性测试	集成期间的存取控制	保存安全级别
8	方法论	按系统开发方法论定义需求	按系统设计方法论设计系统	按编程方法论编写程序	按测试方法论执行测试	在产品环境中集成系统	按系统维护方法论维护系统
9	正确性	定义功能规格说明	设计符合需求	程序符合设计	功能测试	程序和数据安装正确	修改需求
10	容易使用	定义可用性规格说明	系统设计要便于实现可用性需求	程序编写符合易用性设计要求，并进行了优化	人工支持测试	传播可用性指令	保存容易使用
11	可维护	决定可维护的规格说明	设计是可维护的	程序是可维护的	检查	文档齐全	保存可维性
12	可移植	决定可移植的要求	设计是可移植的	程序符合设计	程序符合设计灾难性测试	文档齐全	保存可移植性
13	耦合	定义系统间的接口	设计考虑接口需求	程序符合接口设计说明	功能和回归测试	调整接口	保证正确的接口
14	性能	建立性能准则	保证实际要达到这些准则	程序的实际和实现达到这些准则	符合性测试	监控集成性能	保存性能级别
15	容易操作	定义操作要求	把要求传递给操作	开发操作过程	操作测试	实现操作过程	修改操作过程

计算机系统的风险表现为：产生不正确的结果、系统接受未授权的事务、破坏计算机文件的完整性、不能重新构造处理、破坏处理的连续性、向用户提供的服务将降低到不可接受的程度、将危及系统的安全、结果不可靠、系统难于使用、程序难于维护、不能移植到其他计算机软硬件环境、不可接受的性能级别以及系统难以操作等。

3. 测试计划

1）常见问题

在制定测试计划时我们可能经常遇到下面的问题：

（1）测试计划经常是等到开发后期才开始实行，使得没有时间有效地执行计划。

（2）测试计划的组织者可能缺乏对特殊应用软件测试经验（如嵌入式软件）。

（3）测试的难度和复杂性可能太大，没有自动化工具，很难计划和控制。

2）确定测试策略的因素

在确定测试策略时要考虑以下几个方面：

（1）要使用的测试技术和工具。

（2）测试完成标准。

（3）影响资源分配的因素（如外部接口出现故障、物理设备损坏以及安全受到威胁等）。

测试计划最关键的一步就是将软件分解成单元，写成测试需求。测试需求有很多分类方法，最普通的一种就是按照商业功能分类。把软件分解成单元有几个好处：

① 测试需求是测试设计和开发测试用例的基础，分成单元可以更好地进行设计；

② 详细的测试需求是用来衡量测试覆盖率的重要指标；

③ 测试需求包括各种测试实际所要做的工作，以及所需资源。

3）测试的工作量

测试计划中估计测试工作量一般要从如下几个方面进行综合考虑。

（1）效率假设：测试队伍的工作效率。对于功能测试，这主要依赖于应用的复杂性，如窗口的个数，每个窗口中的动作数目。对容量测试，主要依赖于建立测试所需数据的工作量大小。

（2）测试假设：为了验证一个测试需求所需测试动作数目。

（3）应用的维数：应用的复杂度指标。例如要加入一个记录，测试需求的维数就是这个记录中域的数目。

（4）所处测试周期的阶段：有些阶段主要工作是设计，有些阶段主要是测试执行。

（5）确定测试资源，如硬件和软件环境以及测试工具的系统资源。当然最重要的是人力资源，包括测试项目负责人、测试分析员、测试设计员、测试程序员、测试员、测试系统管理者以及配置管理员等。这些工作人员的职责见表3-3。

表3-3 软件测试人员配备情况

工作角色	具体职责
测试项目负责人	管理监督测试项目，提供技术指导，获取适当的资源，制定基线，技术协调，负责项目的安全保密和质量管理
测试分析员	确定测试计划、测试内容、测试方法、测试数据生成方法、测试（软、硬件）环境、测试工具，评价测试工作的有效性
测试设计员	设计测试用例、确定测试用例的优先级，建立测试环境
测试程序员	编写测试辅助软件
测试员	执行测试、记录测试结果

续表

工 作 角 色	具 体 职 责
测试系统管理员	对测试环境和资产进行管理和维护
配置管理员	设置、管理和维护测试配置管理数据库
注 1：当软件的供方实施测试时，配置管理员由软件开发项目的配置管理员承担；当独立的测试组织实施测试时，应配备测试活动的配置管理员。 注 2：一个人可承担多个角色的工作，一个角色可由多个人承担。	

测试计划按国家标准或行业标准规定的格式和内容编写。

4. 测试种类/技术

软件生命周期中所执行的各类测试如图 3-3 所示。

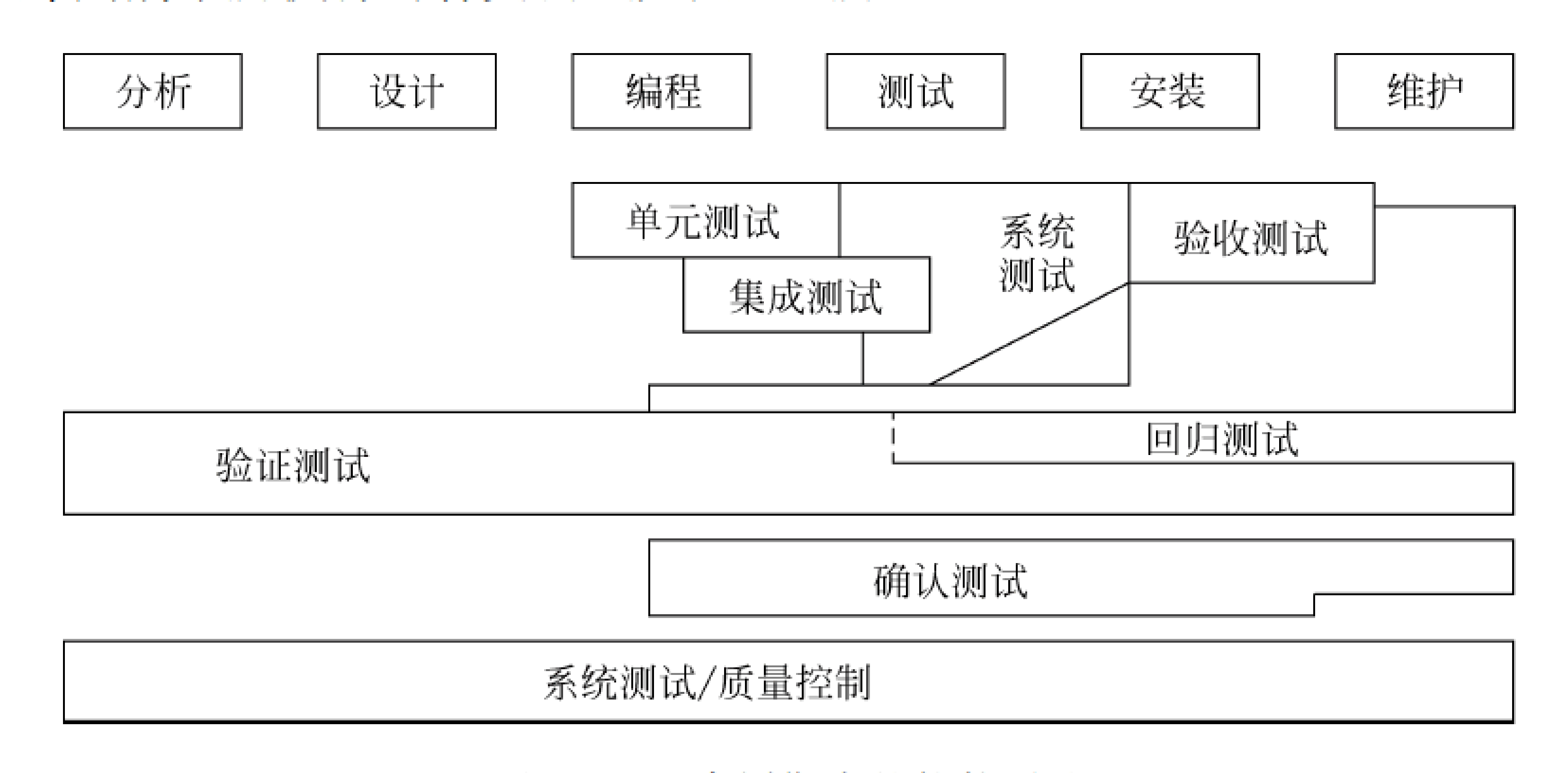


图 3-3　生命周期中的软件测试

下面是软件生命周期中的各类软件测试的定义和概念。

(1) 质量控制(Quality Control)：

① 决定软件产品正确性的过程和动作。

② 一组功能基线，保证产品符合标准/需求所做的工作。

(2) 缺陷(Defect)：

① 偏离规格说明，有三种表现形式(遗漏、错误和多余)。

② 用户不满意的任何事情，不管是否在规格说明书中规定。

(3) 验证(Verification)：在整个软件生命周期中的全部质量控制活动，确保交付的中间产品符合输入规格说明。

(4) 确认(Validation)：软件生命周期中的测试阶段，保证最终产品符合规格说明。

(5) 静态测试：在系统编码之前进行的验证。

(6) 动态测试：在系统编码之后进行的验证和确认。

(7) 单元测试：对单一、独立的模块或编码单元进行的测试。

(8) 集成测试：对一组模块进行的测试，确保模块之间的数据和控制能正常地传递。

(9) 系统测试：

① 一个预先确定的测试组合，当执行成功时，系统符合需求(即确认系统开发正确)；

② 与单元测试不同的各种更高等级测试类型的通用术语。

(10) 验收测试：保证系统符合最终用户要求的测试。

(11) 回归测试：在系统改变后进行的测试，以确保不希望的变化被引入到系统。

(12) 功能测试：认为系统应该做什么的业务需求测试。
(13) 结构测试：确认系统是如何实现的系统结构测试。
(14) “黑盒”测试：数据驱动的、基于外部规格说明而不需了解系统是如何构造的测试。
(15) “白盒”测试：逻辑驱动的、基于编码内部的结构和逻辑的测试。

5. 测试的准入准出条件

1) 准入条件

进入软件测试生命周期一般应具有下列条件，即我们说的测试准入条件：
(1) 具有测试合同(或项目计划)。
(2) 具有软件测试所需的各种文档。
(3) 所提交的被测软件受控。
(4) 软件源代码正确通过编译或汇编。
(5) 能够从一开始介入被测软件的开发周期。

2) 准出条件

软件测试工作结束一般应达到下列要求，即我们说的测试准出条件：
(1) 已按要求完成了合同(或项目计划)所规定的软件测试任务。
(2) 实际测试过程遵循了原定的软件测试计划和软件测试说明。
(3) 客观、详细地记录了软件测试过程和软件测试中发现的所有问题。
(4) 软件测试文档齐全、符合规范。
(5) 软件测试的全过程自始至终在控制下进行。
(6) 软件测试中的问题或异常有合理解释或正确有效的处理。
(7) 软件测试工作通过了测试评审。
(8) 全部测试软件、被测软件、测试支持软件和评审结果已纳入配置管理。

3.1.3 基于风险的软件测试方法

风险可以定义为事件、危险、威胁或情况等发生的可能性以及由此产生的不可预料的后果，即一个潜在的问题。

风险级别由出现不确定事件的可能性和出现后所产生的影响(事件引发的不好的结果，即严重性)两个方面来决定。

在软件测试中，由于测试团队需要在时间、成本和质量等各个方面进行平衡和协调，使得我们无法做到穷尽测试，加上测试时间以及测试资源的限制(还要考虑测试人员的水平因素)，我们很难达到理想的测试目标：使被测软件“零”缺陷。这样，软件就可能存在缺陷和质量问题，由此带来了软件存在着应用上的风险。如故障频发的软件交付使用、软件/硬件对个人或公司造成潜在损害、劣质的软件特性(如功能性、可靠性、可用性和性能)、低劣的数据(如数据迁移问题、数据转换问题、数据传输问题、违反数据标准问题)、软件没有实现既定的功能等。风险的存在也会导致用户或者利益相关者对软件质量或项目成功的信心不足。

1. 什么是基于风险的软件测试

基于风险的软件测试(Risk-Based software Testing，RBT)是指首先评估被测软件的风险，然后根据不同的风险采用不同的测试力度。通常的方法是：

① 列出一个风险的列表。
② 对每个风险进行分析和评估，确定风险级别。
③ 考察每项风险的测试。

④ 当风险消失而新的风险出现的时候，调整测试策略。

现在业界通常对风险进行评估的做法是对每个功能点从业务和技术上考察。业务上是指这项功能失效，对系统的影响。从技术上考察是指实现这个功能的技术难度大不大，是移植的还是新研发的？一般将此两项称为重要性和概率，分别赋予 1 到 5 的权值，5 为最大可能或最重要。

对于重要性为 5、概率为 4 的一个功能点，其乘积为 20，这是一个高风险。对于高风险，就应该用充足的时间、充足的人员来进行测试。

在基于风险的软件测试中，需要解决的主要问题包括确定测试的优先级、选择测试的重点、配置测试的资源、分析和评估测试的有效性等。要有效地选择测试重点和测试优先级，风险测试将测试活动和测试任务根据风险划分优先等级，将测试资源分配在高风险的部分。

这种基于风险的软件测试方法目前得到了广泛的关注，很多机构在他们针对基础级别和进阶级别的测试认证中，将它认定为一种重要的测试方式。

2. 基于风险的软件测试所能解决的问题

基于风险的测试作为软件测试的一种有效方法，可以解决测试过程中面临的一些问题。

1）测试团队面临的问题是测试任务的时间压力

很少有测试项目可以获得足够的时间进行充分的测试。相反，测试一般都是有时间限制的，例如：项目具体里程碑时间、客户或者用户要求产品提交的时间等。基于风险的测试可以提供一种方法，对测试用例和测试任务进行优先级排列。测试的时间限制，其面临的挑战实际就是确定测试的覆盖率。通过基于风险的测试，可以从几乎无限的测试中选择重要的和风险高的测试来开展，从而降低风险和尽快提高质量，提高对产品的信心。测试的时间压力不仅仅存在于测试实现和执行阶段，同样也存在于测试分析和设计阶段。通过基于风险的测试，可以在早期将测试工作量放在高风险的地方，同时可以告知利益相关者这样做可能存在的风险。

2）测试团队经常面临的问题是系统需求质量低下或不完整

通过召集利益相关者讨论哪些是需要测试的、哪些是不需要测试的、测试的深度是多少等问题。基于风险的测试可用来识别需求规格说明中存在的不足。基于风险的测试也可以帮助其他利益相关者认识到测试在确定测试范围和测试深度方面面临的挑战，有助于项目团队成员之间更好地理解和沟通。

3）在项目测试的后期。例如，完成测试执行之后，测试团队需要提供相关的信息给其他的利益相关者，以帮助做出合适的决定

基于风险的测试可以允许测试团队和其他利益相关者一起，根据剩余的风险确定一个可接受的风险级别。而不是仅仅依赖于其他一些不充分的度量，例如缺陷数目、测试用例执行数目等。

3. 基于风险的软件测试的活动实践

有效地应用基于风险的软件测试可以较好地指导软件测试活动的开展，更好地使软件测试活动在时间、成本、质量等方面进行平衡，从而提高测试质量、降低测试成本、缩短测试时间等。下面是基于风险的软件测试方法。

1）确定测试优先级

根据测试风险的分析和评估得到风险分布，确定测试的优先级（风险级别分析也适用于测试的设计和测试实现等阶段，即通过风险分析，确定测试设计和测试实现的优先级）。测试风险的分析基于两个方面：发生的可能性和发生的严重程度。其中，风险发生的可能性主要是从技术方面考虑；而风险发生的严重程度主要是从客户或者用户的角度考虑。

2）确定测试完备性

前面提到的一个假设条件：并不是所有的测试对项目而言是同等重要的。同样的道理，并不需要对测试对象的不同内容进行同等重要的测试。例如，最重要或者风险最大的模块/对象需要进行更加彻底、更加全面的测试。而对于风险比较小、优先级低的模块/对象，可以进行简单测试。对于优先级最低的对象，在时间和成本等不允许的时候，甚至不进行测试。

3）确定测试资源分配

根据测试风险的分析和测试优先级的评估，将经验丰富和技术能力丰富的测试人员（不管是设计人员、实现人员，还是执行人员）放在最重要的模块或测试对象中，以达到最佳效果：①设计更加完善、完备和准确的测试用例；②实现高质量的测试用例脚本和代码；③更加高效地发现测试对象中的缺陷。

4）监控测试进度

根据测试风险的分析和评估，得到测试的优先级和测试重点。接下来，可以根据风险的分布对测试进度进行汇报和控制。例如：测试经理可以根据测试工作的侧重点、测试进度协调人力资源和测试环境的分配，将测试的资源放在最重要的部分。

5）加速测试信心提升

依据测试风险分析和评估得到的测试优先级和测试重点，可以更好、更快地提供产品或者被测系统在质量方面的信心。对被测对象的质量，根据不同的测试策略，得到不同的信心演变过程。

策略1：随机执行测试用例，不分优先级和测试重点，被测系统质量信心的递增是随着测试完成率的递增而线性增加的。

策略2：先执行低复杂度的测试，因此，测试完成率增加很快，但是相应的被测对象质量的信心却增加很慢。而对于高风险（如测试难度较大的大容量用户数据模拟测试）的区域，很可能放在测试的后期进行。

策略3：基于风险的测试，将高风险区域首先进行测试，尽管测试完成率增加比较慢，但是对被测对象质量的信心却增加很快。

3.2 生命周期各个阶段的测试要求

全生命周期中软件测试的最终要求是：①保证软件系统在全生命周期中每个阶段的正确性，验证在整个软件开发周期中各个阶段的软件质量是否合格；②保证最终系统符合用户的要求和需求，验证最终交付给用户的系统是否满足用户需要、符合其需求；③用样本测试数据检查系统的行为特性；④测试的最终目的是确保最终交给用户的产品的功能符合用户的需求，原则是把尽可能多的问题在产品交给用户之前发现并改正。为此，要努力保证生命周期中每个阶段的正确性，使其满足阶段出口的要求。

3.2.1 需求阶段测试

据软件工程统计结果发现50%以上的系统错误是由于错误需求或缺少需求导致的，在需求上发生错误将导致相互纠缠和重复劳动，因而测试费用的80%是花在需求错误的追踪上。

需求测试贯穿了整个软件开发周期，通过需求测试可以知道软件测试的各个阶段，帮助我们设计测试过程、安排测试计划、编写测试用例以及确认测试结果等。有一个正确的需求分析，则大部分缺陷将不会进入到设计和编码阶段，测试所需的费用自然会大大减少，因此，需求

阶段测试的所有花费都是值得的。

1. 需求阶段测试的目标

简单来说，需求阶段测试的目标就是保证需求分析的正确性和充分性。具体地说，需求阶段测试的目标则是保证需求正确反映出用户的需要，需求已经被定义和文档化，项目的花费和收益成正比，需求的控制被明确，有合理的流程可以遵循，有合理的方法可供选择。

2. 需求阶段的测试要素分析

需求阶段的测试要素分析以表3-2为基础，包含的内容有：

① 需求的设计是否遵循了已定义的方法。

② 提交了已定义的功能说明。

③ 定义了系统界面。

④ 已经估计了性能标准。

⑤ 容忍度被预先估计。

⑥ 预先定义了权限规则。

⑦ 需求中预先定义了文件完整性。

⑧ 预先定义了需求的变更流程。

⑨ 预先定义了失败的影响。

3. 需求阶段的测试活动

在需求阶段测试中，需要建立风险列表，进行风险分析和检查，以此确定项目的风险；并且要建立控制目标，确保有足够的控制力度来保证软件项目的开发和测试。在彻底地分析需求的充分性后，生成基础的测试用例。澄清和确定哪些需求是可测试的，舍去含糊的、不可测试的需求，建立产品的测试需求和确认需求。

3.2.2 设计阶段测试

在设计阶段，设计人员需要根据需求分析详细定义要交付的产品——硬件和软件的需求、操作手册说明书、数据保留的策略、输入/输出说明、过程说明、控制说明、系统流程图等。而测试的任务是对设计进行评审，分析测试要素，给测试要素打分，当需求分析发生改变，设计文档也要修改，测试要对修改的部分进行检查，以保证设计和需求的一致性。

1. 设计阶段的测试活动

设计阶段包括概要设计和详细设计。在概要设计阶段，测试人员应阐述测试方法和测试评估准则，编写测试计划，组织成立一个独立的测试小组，安排具有里程碑的测试日程；在详细设计阶段，测试人员要开发或获取确认支持工具，生成功能测试数据和测试用例，以此来检查设计中的遗漏情况、错误逻辑、模块接口不匹配、数据结构不合理、错误I/O假定、用户界面不充分等。

2. 设计阶段的评审

设计阶段的评审是对设计的完整性和正确性进行正式的评价。在对设计进行评审之前，要为评审分配足够的时间，成立评审组，并对组员进行培训；在评审时，要通报项目组，和项目组一起进行评审，并且只对文档进行评审；最后，要将评审的结果写成正式报告。

3. 设计阶段工具的应用

在设计阶段使用静态和动态测试工具测试系统的结构。评分工具和设计评审工具是广泛使用的两种测试工具，评分是标识风险的一种工具，根据得分的结果确定系统的风险程度；而设计评审是对实际阶段处理的完整性进行正式的评价，它是测试设计规格说明的工具，风险越

高，设计评审越详细。

另外，可利用评分工具对测试要素进行分析，给测试要素打分，如：是否设计了对数据完整性的控制？是否设计了权限规则？是否设计了对文件完整性的控制？是否设计了审计追踪？是否设计了发生意外情况时的计划？是否设计了如何达到服务水平的方法？是否定义了权限流程？是否定义了完整的方法学？是否设计了保证需求一致性的方法？是否进行了可用性的设计？设计是否是可维护的、简单的？交互界面设计是否完毕？是否定义了成功的标准？

上述评分过程需要同实际操作者沟通。

3.2.3 编码阶段测试

在编码阶段，测试需要解决的首要问题是编码是否和设计一致；其次是系统是否可维护，系统的规格说明是否正确地实现，编码是否按照既有的标准进行，是否有充分的评价测试计划的可执行程序，程序是否提供了足够的文档资料，程序内部是否有足够的注释等。在测试完成后，要形成下列输出：编码说明书、程序文档、计算机程序列表、可执行的程序、程序流程图、操作介绍和单元测试结果等。

在编码阶段已经开发了很多的测试工具，例如支持程序走查和检查的代码静态分析工具和支持单元“黑盒”测试和单元“白盒”测试的动态测试工具。在编码阶段的测试活动中，有几个方面是需要特别关注的：

① 完成对数据和文件完整性的控制。

② 定义完毕授权的规则。

③ 实现审计追踪。

④ 规划出意外情况发生后的处理计划。

⑤ 编码工作是依据规定的方法完成的(这样易于测试和维护工作的进行)。

⑥ 编码与设计相一致(包括编码的正确性、可用性、间接性和耦合性)。

⑦ 代码是可维护的(代码的维护性在一定程度上决定了项目维护的难易程度)。

⑧ 在性能上定义出程序成功的标准。

3.2.4 测试阶段

测试阶段就是传统软件工程中的软件测试。在全生命周期软件测试方法中，由于在需求、设计、编码阶段都进行了测试，因此测试阶段的问题相对传统的软件测试中的问题要少一些。在测试阶段要进行第三方的正式确认测试，检验所开发的系统是否能按照用户提出的要求运行。在测试阶段要使得用户能成功地安装一个新的应用系统来进行测试。

1. 典型的测试类型

典型的测试类型如下。

1) 手册与文档测试

测试软件的操作说明文档是否全面、正确、简单和满足标准，即测试软件文档的可用性。

2) 一致性测试

测试软件的授权、安全性和性能等是否达到需求分析中的要求。

3) 符合性测试

验证软件系统与相应标准的符合程度。如授权规则是否正确实现，安全方法是否合适，是否按照相应的标准、指南、规程执行测试等。

4）功能测试

运行部分或全部系统，确认用户的需求被满足。这包括可靠性、文件完整性、审计追踪、功能正确性、互连等项测试，检验系统在各种环境和重复的事务条件下能否正确地执行系统的需求，控制计算机文件的完整性，追踪一个原始事务到总的控制，按用户规定的需求测试应用功能，与其他应用系统能否正确地通信。

5）覆盖性测试

检验软件代码各个语句及分支等是否全部执行到。

6）性能测试

通过测量响应时间、CPU 使用和其他量化的操作特征，评估软件系统的性能指标。

7）压力测试

以大信息量的数据进行输入，测试软件的性能。但是这是一个很昂贵的测试，应根据需要来选择。但是在线系统必须要进行压力测试。

8）强度测试

将系统置于强度下进行验收测试，测试系统对极端条件的反应，标识软件的薄弱点，指出系统能够经受的正常的工作量。

9）操作测试

在没有开发人员的指导和帮助情况下，由操作人员进行测试，以评估操作命令的完整性和系统是否容易操作。

10）恢复测试

故意使系统失败，测试人工和自动的恢复过程。

2. 测试用例

设计测试用例的输入数据时，要包含合法和非法的输入（要尝试将测试数据违反程序的规则进行输入），也要描述运行测试用例的预期结果。测试用例需要包含的一些基本信息有标识符（测试设计过程说明和测试程序说明引用的唯一标识符）、输入说明（列举软件执行测试用例的所有输入内容或条件）、输出要求（描述运行测试用例预期的结果）、环境要求（执行测试用例必要的硬件、软件等）、特殊过程要求（描述运行测试必须达到的特殊要求）以及用例之间的依赖性（当一个测试用例依赖于其他测试用例，或者受其他用例影响时，需在此说明）。

3. 测试报告

在测试阶段开始之前，测试人员要参考前期的测试结果和第三方测试反馈（例如计算机操作人员）准备测试阶段的测试计划和测试用例，在完成测试时要生成正式的测试总结报告。生成测试报告的目的是要表示出目前项目的实际情况，给出系统的操作性的评价，为软件的产品化提供一个参考。

测试总结报告的内容有测试结果数据、测试任务、测试集合和测试事件的描述、当前软件状态的描述、各个阶段的项目测试总结等。

3.2.5　安装阶段测试

在进行安装测试时要保证被测试系统没有问题，校验产品文件的完整性。安装要遵循一定的方法和步骤，注重对程序安装的正确性和完整性进行核对。如果安装失败，系统要有相应的解决方案。最后也是最重要的是要保证系统综合的性能达到了用户要求。

1. 安装测试的基本要求

在安装过程中，要根据系统安装手册制定好安装计划，确定好安装流程图，准备好安装文

件和程序清单，给出安装测试的预期结果，并对安装过程中的各种可能发生的结果进行说明和准备，将程序运行的软硬件要求放入产品说明中。同时要检查系统的用户手册和操作手册，看是否可用。

2. 安装测试工作

安装过程中我们要进行的工作有：

(1) 对程序安装的正确性和完整性进行核对。

(2) 校验产品文件的完整性。

(3) 安装的审查、追踪被记录。

(4) 安装之前，该系统已经被证实没有问题。

(5) 如果安装失败，系统有相应的解决方案。

(6) 安装过程，进行了权限控制(安全性)。

(7) 安装遵循一定的方法和步骤。

(8) 需要的配套程序和数据已经放进了产品中。

(9) 已交付使用说明。

(10) 相关文件已经完成(可维护性)。

(11) 接口已经被合理调整(耦合性)。

(12) 综合的性能达到了用户要求。

3.2.6 验收阶段测试

软件验收的流程如下。

1. 定义用户角色

定义用户角色也就是确定软件的用户范围。

2. 定义验收标准

验收标准包括功能、性能、接口质量、过载后的软件质量、软件的安全性和稳定性等方面的标准。

3. 编制验收计划

验收计划包含项目描述、用户职责描述、验收活动描述、验收项的评审和最终的验收测试步骤。

4. 执行验收计划

按照验收计划进行测试和评审。

5. 填写验收结论

验收结束后，填写得出的结论，验收问题必须在进入下一个活动之前被接受和更改。

3.2.7 维护阶段

软件交付使用后的阶段，称为维护阶段。

维护阶段指软件维护阶段的工作重点是测试和培训。

1. 维护阶段中的测试要求

由于软件产品的特殊性，测试过后，并不代表软件没有错误，只能说有些错误还没有被发现，因此在软件交付使用后，仍旧需要对其进行维护。维护人员需要开发一些测试用例，预先发现一些问题；并且要能够根据运行情况的变化和用户的反馈对软件做适当的修正。

2. 维护阶段中的培训要求

在软件交付使用的同时，也要制定培训计划，编写培训材料。培训计划包括对系统进行概览，对系统假定的一些错误给出处理的方法。培训材料包括用户使用方法、对错误列表上的问题给出解释、对输入数据进行解释等。

3.3　支持生命周期软件测试的工具

目前，能够比较好地支持生命周期软件测试的工具主要有微软的 VS 2012、IBM Rational 的一整套自动化测试工具、HP ALM 及配套测试工具，以及 NSEsoftware、LLC 的 Panorama++。为了较好地揭示生命周期软件测试的概念，这里主要以 Panorama++ 和 HP ALM 为例进行工具介绍。

3.3.1　全生命周期质量管理平台 Panorama++

Panorama++作为一个完整的软件工程与量化质量测评管理系统，支持软件开发的整个生命周期，如图 3-4 所示。

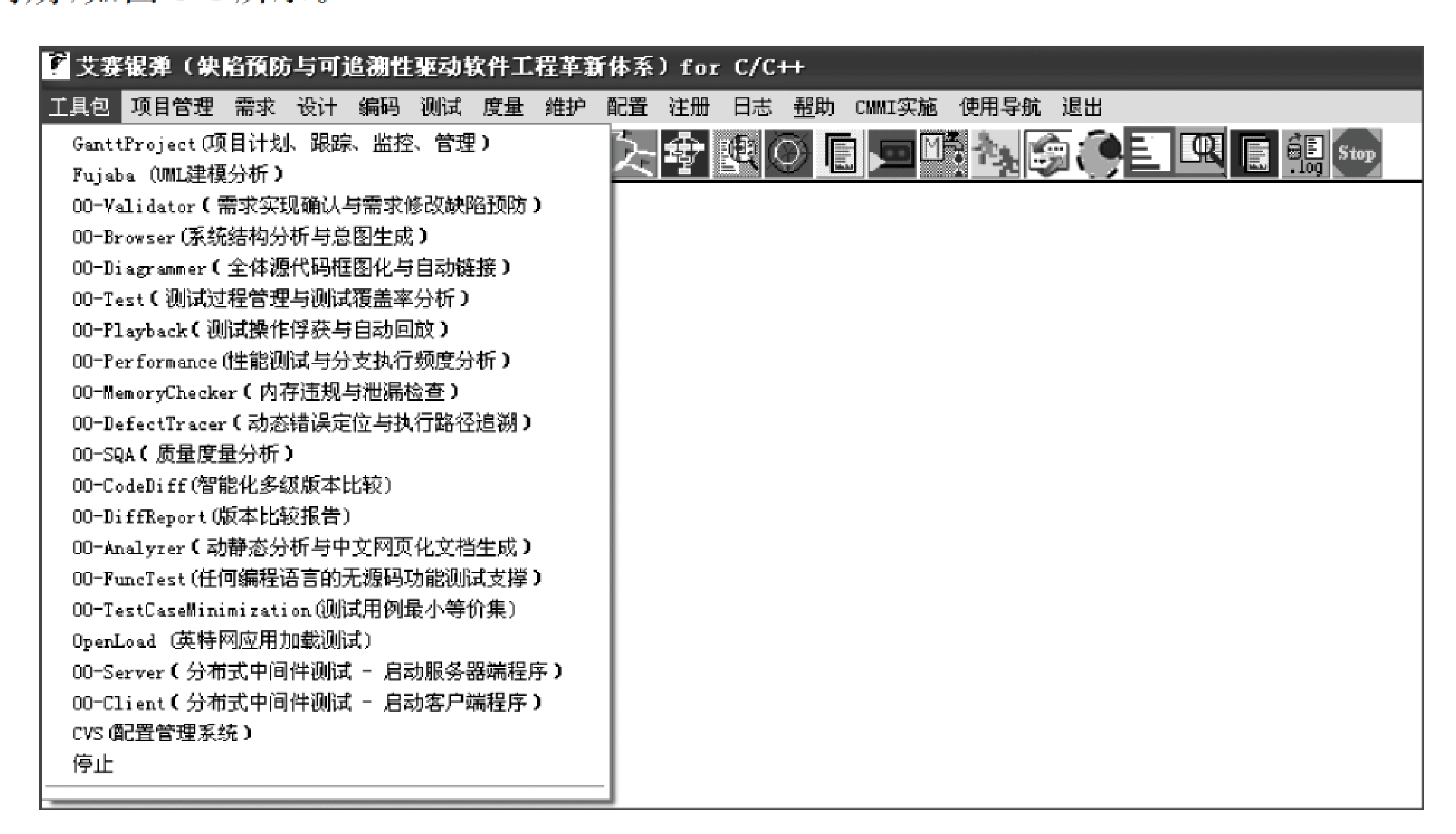


图 3-4　Panorama++工作界面

（1）项目管理支撑：支持立项预估、规划、监控、结项等。

（2）需求阶段支撑：支持 UML 建模分析、管理以及设计和源码间的自动追溯查错等。

（3）设计阶段支撑：支持大型系统的快速分层设计、源码框架生成、逆向工程。

（4）编码阶段支撑：支持增量式、高一致性与低风险编程。

（5）测试阶段支撑：能够支持测试阶段的各种测试活动和测试方法。包括：

① 基于复杂度分析的测试规划；

② 基于整个被测软件的全部源代码的框图化与自动链接的源代码审议和走查支撑；

③ 界面功能测试(采用线性脚本的动作自动俘获与回放，有无源代码均可，后者与所使用的计算机语言种类无关，前者则可与“白盒”结构测试无缝地相结合)；

④ 性能(动态用时分配)测试分析包括各程序分支的执行频度分析；

⑤ “白盒”结构测试，支持美国和欧洲航空航天最高质量标准 RTCA/DO178B-LEVEL A 的 MC/DC(修改条件/判断覆盖)测试覆盖率分析；

⑥ 因特网应用程序的加载测试、分布式中间件与服务器端和客户端的应用程序的配对测试支撑；

⑦ 内存泄漏与违规使用检测、高效率测试用例设计支撑、测试用例有效性分析与测试用例最小等效集的自动生成。

(6) 度量与分析支撑：面向对象的个性化度量项的选择与度量标准的设定支持，动态与静态质量数据的自动收集、自动分析以及多形式的分析结果彩图显示。

(7) 维护阶段支撑：动态运行错误自动定位、错误执行路径追溯、高一致性源码修改支撑、高一致性数据修改支撑。

(8) 配置支撑：版本维护与管理、智能化多级版本比较等。

Panorama++采用以预防错误发生、杜绝错误传递为核心的设计理念，实现了软件系统需求、设计、编码、测试、维护、文档多环节自动相互追溯、精确定位图形化显示，从而也实现了软件全生命周期的测试。

如果没有一个需求及实现该需求的源码间的自动追溯工具，就很难判断功能测试是否充分，很难判断一个需求是否被完全实现，很难确定源码中的一个函数是否真有必要存在于该系统——也许它来自其他已经删去的需求；很难找出其不一致性，也不利于需求和源码的修改维护——一个模块的修改可能关系到一个以上的需求，遗漏了就会产生严重的后果。Panorama++的自动追溯精确定位功能，可以使得软件开发的每个阶段均有测试活动的参与，可以轻松面对系统开发环节中的任意变化，如图 3-5 所示。

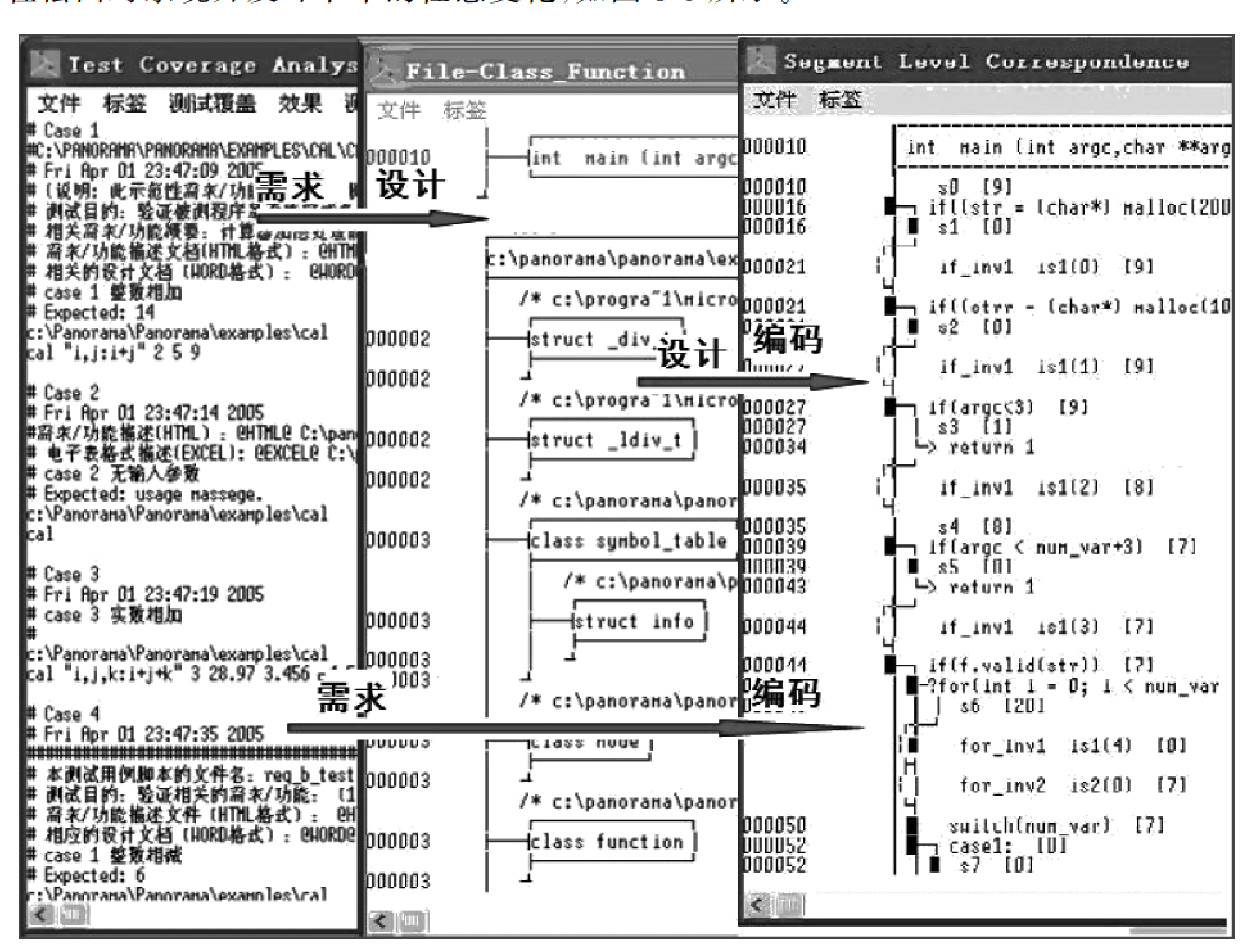
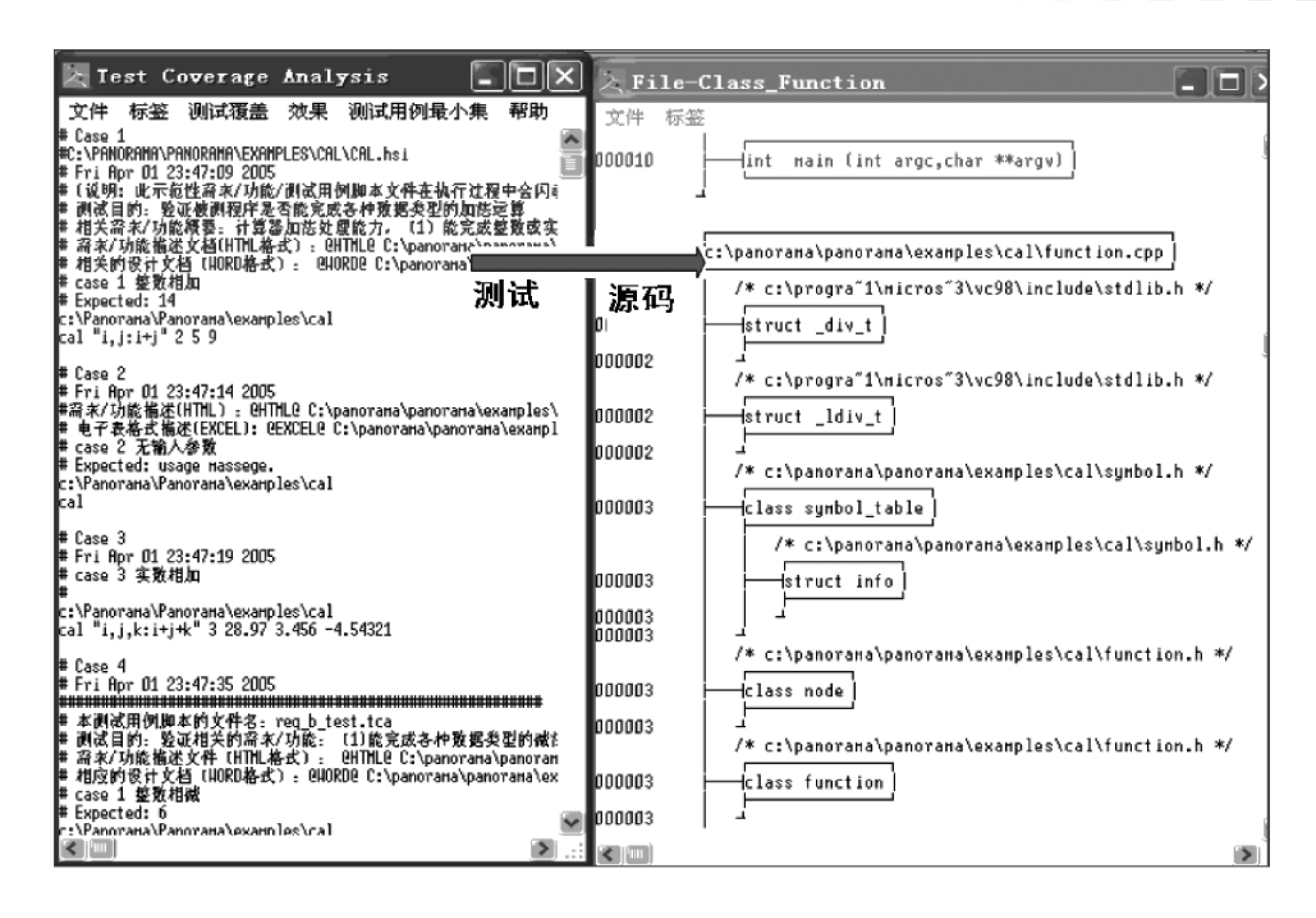


图 3-5 需求、设计、编码和测试之间的自动追溯



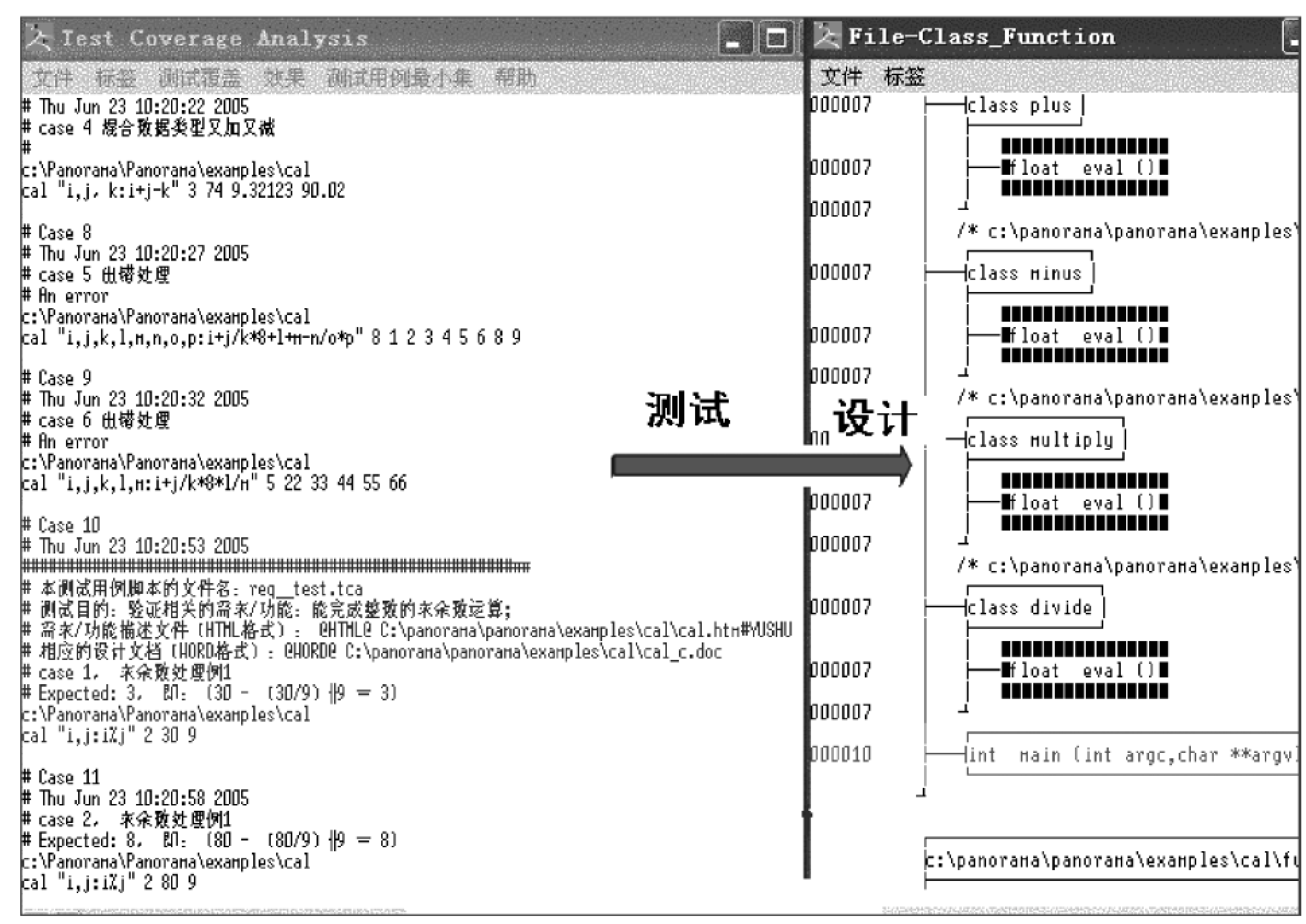


图 3-5　（续）

3.3.2　应用生命周期管理系统 HP ALM11

HP ALM11 是一个以任务为导向的系统，可在应用交付过程中支持各参与方，并与主要开发工具相整合。该方案实现了团队内和不同团队间的工作流程自动化，强化并加速了应用

生命周期管理和各阶段的测试。

1. HP ALM11 能为客户带来的好处

1）管理方面

(1) HP ALM11 项目规划及跟踪(ALM11 Project Planning and Tracking)，建立了发布标准并在整个流程中基于实时监测来管理发布进程及准备情况。

(2) 需求、开发及质量工具间的三路追踪，实现了对应用变化的快速组内分析及执行。

(3) 通过惠普敏捷加速器 4.0(HP Agile Accelerator 4.0)，灵活地按项目类型(瀑布式项目、定制项目、敏捷项目)支持优化交付方式。

(4) 减少因应用故障导致的业务风险，这些应用故障来自由混合以及富互联网应用引起的在功能、性能和安全方面的缺陷。

(5) 通过易于发现、重复利用以及分享关键应用工具(包括需求、测试及缺陷检测)，降低成本并缩短交付时间。

2）测试实施方面

HP ALM11 能简化和自动化应用质量和性能验证，从而降低运营成本，并将更多的资源投入到新应用及服务中。它能带来如下好处。

(1) 借助 HP Sprinter，通过自动化手动测试(如数据创建，以及在多环境下进行的重复手动测试)来加速应用部署。

(2) 通过 HP TruClient 改善测试创建，TruClient 是 HP Load Runner 11.0 的一部分，主要对应用性能进行测试，无须执行耗时的脚本处理。

(3) 借助 HP Unified Functional Testing 11.0，为复合应用提供单一自动化解决方案，可降低 GUI 和非 GUI 测试应用功能故障。该方案由惠普功能测试(HP Functional Test)和惠普服务测试 11(HP Service Test 11)组成。

2. HP ALM 的主要特性

HP ALM 有如下的主要特性。

1）综合管理

HP ALM 通过横跨整个应用交付流程的公共平台，为不同团队提供了不同工具组合。这些团队包括企业架构师、业务分析员、开发人员、质量保证(QA)专业人员、安全专家及生产团队。HP ALM 为规划、创建及发布高复杂性应用提供了一套完整的视角。该平台提供了可轻松拓展、现代化开放的架构。

2）增强可预见性

HP ALM 中全新的项目规划及追踪解决方案可精确地预测项目进展，从而增强可预见性。在项目经理制定了项目计划、时间点、关键绩效指标(KPI)及每个任务的退出条件后，HP ALM 在应用生命周期中可自动追踪不符合已制定的时间点的活动及关键绩效指标，并向相关负责人发出警告。通过这样一个可信赖的、能够时时监控状态的平台，哪怕最复杂的应用项目都可以轻松管理。

3）更密切的业务合作

HP ALM 使用常见工具创建应用需求，促进了业务、应用开发人员和质量保证专业人员间的交流，这些工具包括：

(1) 业务流程模型真实地展现了工作进程，使业务分析员能创建一整套应用需求，避免重复或疏忽，从而使业务分析员、质量保证人员、开发人员及安全人员可高效合作，以创建满足业务需求的应用。

(2) 丰富的文本编辑器(rich text editor)提供了与微软 Word 相似的数据输入功能,可将应用需求输入 HP ALM 中,这大大加快了业务分析员使用速度。

(3) 定制化的模板及工作流程使业务分析员在企业统一的架构中轻松捕获应用需求,从而避免了重复劳动,并可一致、有效且明确地定义需求,获得高品质应用。

4) 增强协作

IDEs 对于增强应用团队中相关各方的协作非常重要。HP ALM 预先集成在全部主要 IDEs 中,从而使开发人员不必使用其他工具就能直接从工作环境中查看应用需求及缺陷。HP ALM 与微软的 Visual Studio/TFS 和 IBM 的 Eclipse 进行了集成,实现即开即用,可在需求、缺陷检测和源代码间建立可追溯性。

5) 单一的综合平台

HP ALM 平台,助力应用团队管理整个应用周期内的测试,包括监控项目的生命周期状态。因而可使客户改善应用的可预见性、可重复性和质量,同时做好准备应对从业务构想到退出的各种变化。

(1) HP Quality Center 助力客户达成:

① 通过对需求测试及缺陷检测的详细追踪改善应用质量。

② 根据风险级别调整和确定测试工作优先级,实现资源优化。

③ 通过对应用交付流程的实时报告增强可预见性。

④ 通过应用 HP Sprinter,提高手动测试效率并增强创新性。

⑤ 通过一个共享的需求、测试及缺陷检测的中央储存文件来增强可重复性。

(2) 惠普性能中心 11.0(HP Performance Center 11.0)助力客户达成:①通过对需求测试及缺陷检测的详细追踪改善应用性能;②在分散团队中通过增强对 COE(Centers Of Excellence)的支持,实现效率最大化,其中包括版本控制、资产共享和项目分组;③通过包含整个生命周期的应用质量、性能和安全等内容的统一仪表板增进协作性;④通过测试环境下的拓扑型系统基础设施视图增强可视性。

3. 应用程序生命周期管理过程

ALM 能够帮助我们组织和管理应用程序生命周期管理过程的所有阶段,包括定义版本、指定需求、计划测试、执行测试和跟踪缺陷。

使用 ALM 的应用程序生命周期管理过程包括以下阶段。

(1) 指定版本:制定一个发布周期管理计划,更高效地管理应用程序发布和周期。追踪应用程序发布,并根据计划确认发布是否正常。

(2) 指定需求:分析应用程序并确定需求。可以跨多个发布和周期管理需求,并在需求、测试、缺陷之间实现多维追踪。ALM 为需求覆盖和关联到质量评估和商业风险中的缺陷提供实时可见的功能。

(3) 计划测试:创建一个基于需求的测试计划,ALM 为手动和自动测试都提供了知识库。

(4) 执行测试:创建测试集,完成测试运行。ALM 支持健壮测试、功能测试、回归测试、更高级测试。根据计划来执行测试,从而识别和解决问题。

(5) 追踪缺陷:报告在应用程序中检测到的缺陷,跟踪修复进程。分析缺陷和缺陷趋势,帮助做出合理的“执行/不执行”决策。ALM 支持完整的缺陷生命周期——从初始问题检测到修复缺陷以及确认缺陷修复。

在每个阶段,可以通过生成的详细报告和图表来分析数据。

ALM 是一个基于 Web 的工具，用来给基于 Web 的项目创建知识库。它是一个用 J2EE 开发的客户端/服务器组织结构，是一个服务的集合体。

习题

1. 详细说明软件工程生命周期 V 形图的含义。
2. 怎样才能把好软件工程生命周期各个阶段的质量关？
3. 什么是生命周期测试方法？生命周期测试如何开展？
4. 生命周期测试有哪些测试任务？简述测试策略、测试要素及测试风险各自的含义。
5. 举例说明计算机系统的风险表现。简述基于风险的软件测试方法。
6. 如何制定测试计划？在制定测试计划时，应考虑哪些因素？
7. 测试阶段有哪些测试内容？用到哪些技术？
8. 需求阶段、设计阶段、编码阶段、测试阶段、安装阶段、验收阶段及维护阶段需要进行哪些测试？
9. Panorama++ 和惠普 ALM 是如何支持生命周期测试的？

第4章 软件测试分类与分级

软件测试是一项复杂的系统工程,对软件测试进行类别与级别的划分从不同的角度考虑可以有不同的划分方法。对测试进行分类和分级是为了更好地明确测试过程中的测试任务分类和测试过程中存在的不同级别,清楚测试各个阶段和开发各个阶段的对应关系,了解整个测试究竟要完成哪些工作和哪些任务,尽量对测试工作做周全安排,测试任务合理分配,测试资源最佳调度,测试管理科学化。

4.1 软件测试分类

按照全生命周期的软件测试概念,测试对象应该包括软件开发的各个阶段的内容,对于需求和设计等阶段的测试前面已有论述,这里重点讲述编码阶段及后面阶段的软件测试。

对于软件测试,可以从不同的角度进行分类。例如,从是否关心软件内部结构和具体实现的角度划分,软件测试可以分为"白盒"测试、"黑盒"测试和"灰盒"测试;从软件开发的过程的角度,软件测试可以分为单元测试、集成测试、系统测试、验收测试;从是否执行程序的角度划分,软件测试可以分为静态测试和动态测试;从测试执行时是否需要人工干预的角度划分,软件测试可以分为人工测试和自动化测试;从测试实施组织的角度划分,软件测试可分为开发方测试、用户测试(β测试)和第三方测试。

我国GB/T 15532—2008计算机软件测试规范给出了基于计算机软件配置项的软件测试分类方法,下面我们就这种分类方法进行全面介绍。

4.1.1 计算机软件配置项

计算机软件配置项缩写为CSCI,是为独立的配置管理(技术状态管理)而设计的且能满足最终用户要求的一组软件,简称软件配置项。

1. 软件配置项概念

在软件开发过程中,产生的所有信息构成软件配置,它们是代码(源代码和目标代码)、文档(需求文档、技术文档、管理文档等)、报告等(包括软件测试过程中所产生的许许多多的工作成果,例如测试计划、测试用例、测试问题报告、测试总结报告以及自动化测试执行脚本和测试缺陷数据等),它们都应当被保存起来,以便查阅和修改。这些纳入配置管理范畴的工作成果统称为配置项(Configuration Item,CI),每个配置项的主要属性有名称、标识符、文件状态、版本、作者、日期等。

在整个软件生存周期内,软件配置管理控制这些软件配置项的投放和变更,记录并报告配置的状态和变更要求,验证配置的完整性、正确性和一致性。然而,尽管这些变动中绝大部分是合理的,但是在不同的时机做不同的变动,难易程度和影响的结果差别仍旧很大,为了更有

效地控制变更,引入了基线的概念。

2. 基线概念

基线即软件技术状态基线,指需要受到配置管理控制的某个研制阶段终点处的软件成分的技术状态,是已经经过正式审核和同意,是下一步软件开发的基础。任何一个软件配置项,一旦形成文档并审议通过,即成为基线。如果要对已经成为基线的软件配置项进行修改时,必须按照特殊的、正式的过程进行评价,确认其修改;相反,对于未成为基线的软件配置项,可以进行非正式的修改。它的作用是使各阶段工作的划分更加明确,使本来应该是连续的工作可在这些点上断开,以便于检验和肯定阶段成果。每一个基线都是其下一步开发的出发点和参考点。基线确定了元素(配置项)的一个版本,且只确定一个版本。一般情况下,基线一般在指定的里程碑处创建,并与项目中的里程碑保持同步。

3. 软件配置管理

软件配置管理通过协调在同一软件项目中不同人员的软件工作产品来帮助我们减轻这些问题。它涉及:①识别、定义软件配置项并指定基线;②控制软件配置项的修改与发行,记录与报告软件配置项的状态和修改请求;③保证软件配置项的完整性、一致性和正确性;④履行必要的审批手续,控制软件的存储、处理和交付。

若缺乏上述这种控制,在同一软件项目中,不同人员的工作就会发生冲突,结果会导致如下问题:

(1) 同时修改软件的冲突(当两人或两人以上共同开发同一软件时,最后一个人的修改很可能损害前人的工作)。

(2) 共享代码的冲突(当一个人修改了共享代码后,通常不是所有的人都能知道)。

(3) 公共代码的冲突(在一些大型系统中,当修改通用的软件功能时,所有使用这些公共代码的人都必须知道,才不会造成不必要的错误。因此如果缺乏有效的代码管理,就没有办法保证一一找到,并一一提醒所有的使用者)。

在软件开发过程中,开发团队通常已将我们要测试的软件按功能和性能要求合理地分解到了各个软件配置项中,并划分了软件配置项关键等级且形成清单。我们可以基于这些文档作为软件测试和质量控制的依据。

4.1.2 基于 CSCI 的软件测试分类

GB/T 15532—2008 计算机软件测试规范给出的测试类别是单元测试、集成测试、配置项测试(也称软件合格性测试或确认测试)、系统测试、验收测试和回归测试,我们可根据软件的规模、类型、完整性级别选择执行测试类别。对这些测试类别的描述结构包含测试对象和目的、测试的组织和管理、技术要求、测试内容、测试环境、测试方法、测试过程和文档要求。

其中软件配置项测试的对象是软件配置项,其测试目的是检验软件配置项与软件需求规格说明的一致性。对于软件配置测试项的测试要确保其测试工作的独立性——一般由软件的供方组织,由独立于软件开发的人员实施,软件开发人员配合。

软件配置项测试的技术依据是软件需求规格说明(含接口需求规格说明),其测试内容主要依据本教程第 5 章叙述的软件质量模型中包括功能性、可靠性、可用性、效率、维护性和可移植性中的 27 个质量特性及子特性等的可测试项,根据软件需求进行选定。当然可根据被测对象的实际情况进行测试内容的剪裁。

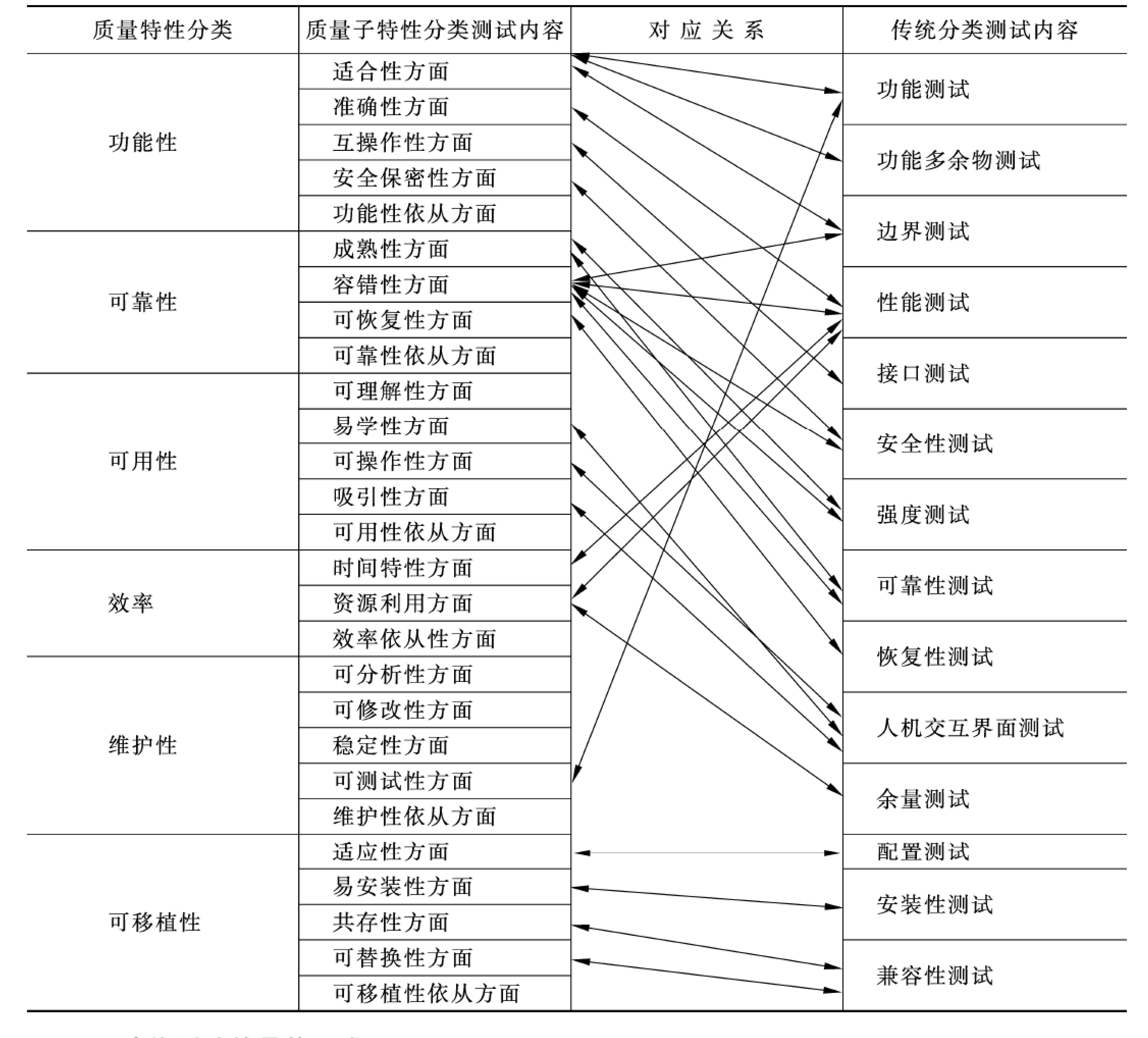
为保证上述测试类别测试的充分性,有必要进行以下种类的测试:功能测试、可靠性测试、性能测试、安全性测试、边界测试、安装性测试、余量测试、恢复性测试、接口测试、功能多余

物测试和强度测试。

对需要支持中文本地化的软件，我们还要进行中文能力测试；对应用软件系统在有条件和有要求的情况下进行应用基准测试。

基于质量特性及质量子特性的配置项测试与上述测试类型的对应关系如表 4-1 所示。可根据这些确定软件测试各个级别中的测试内容。

表 4-1 测试内容不同分类的对应关系



质量特性分类	质量子特性分类测试内容	对应关系	传统分类测试内容
功能性	适合性方面		功能测试
	准确性方面		
	互操作性方面		功能多余物测试
	安全保密性方面		
	功能性依从方面		边界测试
可靠性	成熟性方面		
	容错性方面		性能测试
	可恢复性方面		
	可靠性依从方面		接口测试
可用性	可理解性方面		
	易学性方面		安全性测试
	可操作性方面		
	吸引性方面		强度测试
	可用性依从方面		
效率	时间特性方面		可靠性测试
	资源利用方面		
	效率依从性方面		恢复性测试
维护性	可分析性方面		
	可修改性方面		人机交互界面测试
	稳定性方面		
	可测试性方面		余量测试
	维护性依从方面		
可移植性	适应性方面		配置测试
	易安装性方面		安装性测试
	共存性方面		
	可替换性方面		兼容性测试
	可移植性依从方面		

1. 功能测试的具体要求

功能测试主要对软件需求规格说明中的功能需求进行测试，找出被测软件的实现与需求不一致的地方，确认一致的地方。进行功能测试时，要求：

（1）每一个软件功能必须被一个测试用例或一个被认可的异常所覆盖。

（2）用基本数据类型和数据值测试。

（3）用一系列合理的数据类型和数据值运行，测试超负荷、饱和及其他“最坏情况”的结果。

（4）用假想的数据类型和数据值运行，测试其排斥不规则输入的能力。

（5）每个功能的合法边界值和非法边界值都必须有测试用例专门测试。

2. 性能测试的具体要求

性能测试主要对软件需求规格说明中定义的性能需求进行测试,说明在一定工作负荷和配置条件下,系统的响应时间及处理速度等特性,找出被测软件的实现与性能需求之间的不一致。在进行性能测试时,要求:

(1) 测试程序在获得定量结果时程序计算的精确性。

(2) 测试程序在有速度要求时完成功能的时间。

(3) 测试程序完成功能所能处理的数据量。

(4) 测试程序各部分的协调性,如高速、低速操作的协调。

(5) 测试软件/硬件中的有关因素是否限制了程序的性能。

(6) 测试程序的负载潜力。

(7) 测试程序运行占用空间。

3. 外部接口和人机交互界面测试具体要求

外部接口和人机交互界面测试主要对平台各个服务域提供的应用编程接口、应用程序接口、外部环境接口以及人机交互界面的符合性和可用性进行测试。在进行外部接口和人机交互界面测试时,要求:

(1) 测试所有外部接口,检查接口信息的格式及内容。

(2) 测试所有人机交互界面提供的操作和显示界面,并以非常规操作、误操作、快速操作来检查界面的可靠性,以最终用户为背景检验界面显示的清晰性。

(3) 如果有用户手册或操作手册,应对照手册逐条进行操作和观察。

4. 强度测试具体要求

强度测试必须在预先规定的一个时期内,在软件设计能力的极限状态,进而超出此极限状态下,运行软件的所有功能。

5. 余量测试具体要求

余量测试是测试程序全部存储量、输入输出通道及处理时间的余量是否满足需求规格说明的要求。

6. 可靠性测试具体要求

可靠性测试是在有代表性的环境中,为进行软件可靠性估计而对其进行的功能测试。进行可靠性测试时,要求:

(1) 软件可靠性测试必须按照使用的概率分布随机地选择测试实例。

(2) 必须保证输入覆盖,包括输入域覆盖(即覆盖重要的输入变量值,并且所有被测输入值域的概率之和必须大于软件可靠度要求)、各种使用功能的覆盖、相关输入变量可能性组合的覆盖(以确保相关输入变量的相互影响不会导致软件失效)、设计输入空间与实际输入空间之间区域的覆盖(即不合法输入域的覆盖)。

(3) 被测软件的测试环境(包括硬件配置和软件支撑环境)应和预期的实际使用环境尽可能一致。

(4) 对于可能导致软件运行方式改变的一些边界条件(如堆栈溢出)和环境条件(如系统加电、掉电、电磁干扰等)必须进行针对性测试。

(5) 测试时应记录测试结果、运行时间和判断结果。如果软件失效,还应记录下失效现象和时间。

7. 安全性测试具体要求

安全性测试主要对平台软件配置项的安全性进行测试,说明安全系统是否存在,是否起到

了应有的作用，是否达到了规定的安全级别等。安全性测试的内容包括系统安全评估和系统入侵测试两个部分。系统安全测试主要涉及标识与鉴别、访问控制、审计、特权管理、可信通路、隐通道等；系统入侵测试主要涉及系统脆弱性分析、系统安全漏洞检测等。在进行安全性测试时，要求：

（1）必须进行软件安全性分析，并且在软件需求说明中明确每一个危险状态及导致危险的可能原因，在测试中全面检验软件在这些危险状态下的反应。

（2）对安全性关键的软件部件，必须单独测试，以确认该软件部件满足安全性要求。

（3）对软件设计中用于提高安全性的结构、算法、容错、冗余、中断处理等方案必须进行针对性测试。

（4）测试应尽可能在符合实际使用的条件下进行。

（5）除在正常条件下测试外，应在异常条件下测试软件，以表明不会因可能的单个或多个输入错误而导致不安全状态。例如：

- 必须包含硬件及软件输入故障模式测试。
- 必须包含边界、界外及边界接合部的测试。
- 必须包括0、穿越0以及从两个方向趋近于0的输入值。
- 必须包含在最坏情况配置下的最小和最大输入数据率（以确定系统的固有能力及对这些环境的反应）。
- 操作员接口测试必须包括在安全性关键操作中的操作员错误（以验证安全系统对这些错误的影响）。
- 应测试双工切换、多机替换的正确性和连续性。
- 应测试防止非法进入系统并保护系统数据完整性的能力。

对于测试中发现的缺陷，必须纠正，以消除所有危险或将其风险降到可接受水平。纠正后的软件应在同样的条件下重新测试，以确保已消除危险和不会出现其他危险。

8. 恢复性测试具体要求

对有恢复或重置（RESET）功能的软件，必须验证恢复或重置功能，对每一类导致恢复或重置的情况进行测试。

9. 边界测试具体要求

边界测试是测试程序在输入域（或输出域）、数据结构、状态转换、过程参数、功能界限等的边界或端点情况下的运行状态。

10. 功能多余物测试具体要求

功能多余物测试是验证程序中不存在软件需求中没有指明的功能及功能边界的不适当问题。所有输出都应有意义并在软件需求中指明。

11. 安装性测试具体要求

安装性测试主要对平台软件配置项的可安装性/可卸载性进行测试。安装性测试通过安装/卸载程序或按照安装/卸载规程进行软件配置项的安装/卸载，发现安装/卸载过程的错误，验证软件配置项的可安装性/可卸载性，包括参数装订、程序从介质装入计算机等。

12. 中文能力测试具体要求

中文能力测试主要对平台软件配置项的中文支持能力进行测试，验证软件配置项是否能全面、正确地支持和处理中文。进行中文能力测试时，要求：

1）测试中英文转换后的正文长度变化是否对软件有影响

短消息和命令名经过翻译后可能需要更多空间，这时正文可能会超出菜单或对话框，正文

字符串在内部存储区中也许会溢出，覆盖其他的代码或数据。

2）测试软件是否能完全处理中西文字符集

通常一个典型的字符集包含256个字符，其编号从0～255。其中编号从32～127的字符为ASCII字符，编号从0～31的符号为低端ASCII，编号从128～255的符号为高端ASCII。从技术上来说，任何将0～255的数字与符号联系起来的符号集就是一个代码页。在中文代码页与西文代码页之间，编号从128～255的符号是存在冲突的，对于中文能力测试来说，必须测试在不同的代码页之间切换时高端ASCII字符是否能正确显示。

3）测试对中文是否存在正文过滤现象

有些程序在一个字段中只能接收一定的字符，也许会将高端ASCII字符和各种控制码屏蔽掉，这对英语来说是适当的，但对非英语字符来说可能是错误的。因此，必须在你能输入字符的任何地方测试一个程序是怎样接收和显示所有字符的。

4）测试操作系统语言是否支撑中文

测试通配符、文件名定界符、文件名约定在不同语言的环境下的作用是否正确。

5）测试中文排序规则是否正确

字符排序规则在各个国家都各不相同。按内部代码值排序在中文里没有任何意义。测试程序的排序功能是否符合中文习惯。

6）测试大小写转换功能对中文是否有影响

中文不存在大小写，测试程序的大小写是否只对西文起作用。

13．应用基准测试具体要求

应用基准测试主要对平台软件配置项的综合性能进行测试。应用基准测试通过构造一组能够反映某个领域应用特点的典型应用程序，即面向特定应用领域的应用基准程序，并使之在平台上加以运行，来测试平台软件配置项的综合性能，如软件配置项的适用性、准确性、成熟性、稳定性、时间行为、资源行为、适应性、易学性、易操作性等。

应用基准测试主要用来验证平台软件系统对典型应用的综合支持能力，因此，应用基准程序必须是能够反映应用领域特点的典型，甚至标准的应用程序。

在进行测试时需要注意三点：

① 必须有交办方人员参加计算机软件配置项测试。

② 全部预期结果、测试结果及测试数据应存档保留。

③ 建立独立的测试小组进行计算机软件配置项测试。

上述测试种类共包含13类。事实上，在实际测试中并非在每个测试级别上都要完成上述所有的测试种类，也不是对任何软件都要完成上述所有的测试。究竟必须执行哪些测试，应根据软件的复杂性、关键等级和当前的测试级别选定。例如在某工程中，对此规定如下：

① 单元测试阶段至少完成功能测试、边界测试。

② 集成测试阶段至少完成功能测试、性能测试、余量测试、边界测试和接口测试。

③ 软件配置项测试阶段至少完成功能测试、性能测试、余量测试、边界测试和接口测试。

④ 对于安全关键等级为A、B级的软件还必须完成强度测试、可靠性测试、安全测试和功能多余物测试。

4.2 软件测试分级

对软件测试的要求、目的、关注点、被测对象、工作产品及测试人员不同，相应的软件测试级别划分或分级是不同的。目前大家常关注的软件测试的有关分级大致包括软件生命周期测

试的分级、错误及它对软件测试通过影响的分级、完整性测试的分级、软件测试用例的分级等。通过对软件测试进行有目的的分级，使我们能够有效地控制软件的复杂性，强化测试的针对性或目的性，提高测试管理的科学性，最终确保软件测试的质量。

本节主要介绍软件生命周期的测试分级和软件测试中的错误分级等内容，其他的分级内容在后面章节将有选择地介绍。

4.2.1　软件生命周期的测试分级

我国国标 GB/T 15332—2008 计算机软件测试规范、国军标 GJB 2725A—2001 附加文件——《军用软件测评实验室测评过程和技术能力要求》以及相关行业制定的软件测试规范或标准均是按照软件生命周期对软件测试进行了级别划分。在国标中是以测试类别命名，在国军标中是以测试级别命名。大部分涉及单元测试或组件测试、集成测试、配置项测试（也称软件合格性测试或确认测试）、系统测试和验收测试等分级内容。

针对不同的测试级别，我们应该明确：

① 不同的测试对象。

② 每个测试级别的测试目的。

③ 测试用例设计的基础（所参考的软件产品或测试需求）。

④ 发现的典型缺陷和失效。

⑤ 测试工具的需求和支持。

⑥ 不同的测试技术和方法。

测试级别可以根据软件的规模、类型、安全性关键等级进行选择。

回归测试可出现在上述每个测试级别中，并贯穿于整个软件生命周期。

1. 组件测试

针对单个软件单元的测试都可以称为组件测试（根据开发人员和编程语言不同，软件单元可以是模块、单元、程序或功能）。

1）组件测试方法

在组件测试过程中，经常会用到桩、驱动器、模拟器。这是由于一般被测模块不能形成一个完整的可测试系统，因此在组件测试过程中，需要为测试模块开发驱动器和桩模块。驱动器和桩模块是测试过程中使用的软件，不是软件产品的组成部分，也是需要相关费用的。

组件测试包括功能测试和特定的非功能测试，例如：资源行为测试（内存泄露）、健壮性测试、基于结构的测试（如分支覆盖）等。组件测试的设计输入主要是单元详细规格说明、软件设计规格说明或数据模型等。

在编写代码之前就开始准备测试和自动化测试用例是组件测试常用的一种方法，称为测试驱动的方法或测试驱动开发。

组件测试的任务主要有模块局部数据结构测试、模块参数边界值测试、模块中所有独立执行路径测试以及模块的各条错误处理路径测试等。

当程序代码编写完成并通过评审和编译检查后，便可开始组件测试。对于这个测试级别，测试是在与开发紧密合作的情况下进行的，一般是由开发人员自己来执行组件测试。组件测试主要采用“白盒”测试方法，通常从程序内部结构出发设计测试用例。

2）组件测试需要考虑的因素

组件测试中需要考虑各方面的因素，包括：

（1）检查单元模块接口参数，是组件测试的基础。

(2) 检查局部数据结构,用来保证临时存储在模块内的数据在程序执行过程中的完整、正确。

(3) 在模块中应对每一条独立执行路径进行测试,组件测试的基本任务是保证模块中每条路径至少执行一次。

(4) 比较、判断与控制流常常紧密相关。

(5) 好的软件设计应能预见各种出错条件,并预设各种出错处理路径,出错处理路径同样需要认真测试。

2. 集成测试

集成测试也叫组装测试、联合测试,是一种旨在暴露接口以及集成组件/系统间交互时存在缺陷的测试。是组件测试的逻辑扩展,其关注点是对组件之间的接口进行测试,以及检查与系统其他部分相互作用的测试。如操作系统、文件系统、硬件或系统之间的接口。

1) 集成测试类别划分

根据不同的测试对象规模,可分为组件集成测试和系统集成测试。

组件集成测试对不同的软件组件之间的相互作用进行测试,一般在组件测试之后进行;系统集成测试对不同系统之间的相互作用进行测试,一般在系统测试之后进行。

系统集成测试的范围越大,对缺陷的定位越困难,从而增加了系统的风险。为了降低在软件生命周期后期发现严重缺陷而产生的风险,集成程度应该逐步增加。

2) 集成测试方法与策略

集成测试感兴趣的是模块之间的接口,而不是模块本身的功能,“黑盒”测试和“白盒”测试的方法都可以应用在集成测试上。

集成测试的对象是已经经过组件测试的软件单元,集成测试的依据是软件的概要设计规格说明。

集成测试采用的测试策略是非渐增式集成测试模式和渐增式集成测试模式,具体包括自底向上集成、自顶向下集成、核心系统先行集成、随意集成等。

3) 集成测试内容

集成测试的主要内容是功能性、可靠性、可用性、效率、可维护性和可移植性。

3. 配置项测试

配置项测试的对象是计算机软件配置项。配置项测试可根据软件测评任务书、合同或其他等效文件及软件配置项的重要性、安全性关键等级对如下要求内容进行剪裁,但必须说明理由。配置项测试一般应符合以下技术要求:

(1) 必要时,在高层控制流图中作结构覆盖测试。

(2) 应逐项测试软件需求规格说明规定的配置项的功能、性能等特性。

(3) 配置项的每个特性应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖。

(4) 测试用例的输入应至少包括有效等价类值、无效等价类值和边界数据值。

(5) 应测试配置项的输入输出及其格式。

(6) 应测试人机交互界面提供的操作和现实界面,包括用非常规操作、无操作、快捷操作以及快速操作等来测试界面的可靠性。

(7) 应测试运行条件在边界状态和异常状态下,或在人为设定的状态下,配置项的功能和性能。

(8) 应按软件需求规格的要求,测试配置项的安全性和数据的安全保密性。

(9) 应测试配置项的所有外部输入输出接口(包括和硬件之间的接口)。

(10) 应测试配置项的全部存储、输入输出通道的吞吐能力和处理时间的余量。

(11) 应按照软件需求规格的要求,对配置项的功能、性能进行强度测试。

(12) 应测试设计中用于提高配置项的安全性和可靠性方案,如结构、算法、容错、冗余、中断处理等。

(13) 对安全性关键的配置项,应对其进行安全性分析,明确每一个危险状态和导致危险的可能原因,并对此进行针对性测试。

(14) 对有恢复或重置功能需求的配置项,应测试其恢复或重置功能和平均恢复时间,并对每一类导致恢复或重置的情况进行测试。

(15) 对不同的实际问题应做专门测试。

4. 系统测试

系统测试是将已经集成好的软件系统作为计算机系统的一部分,与计算机系统硬件、某些支持软件、数据和人员等系统元素结合起来,在实际运行环境下对计算机系统进行一系列严格有效的测试。

系统测试对测试环境的要求是在集成测试完成后,系统已经完全组合起来后进行,应该在尽可能和目标运行环境一致的情况下进行。

系统测试的目的是确认整个系统是否满足了系统需求规格说明中的功能和非功能需求,以及满足程度。

常见系统测试包括压力测试、容量测试、性能测试、安全测试、容错测试等。

5. 验收测试

验收测试通常由使用系统的用户来进行测试,目的是确保系统功能、系统的某部分或特定的系统非功能特征满足验收准则,发现缺陷不是验收测试的主要目标。

验收测试的主要测试类型有根据合同的验收测试、用户验收测试、运行(验收)测试、现场测试。

6. 维护测试

维护测试是指软件被市场接受后,在运行一段时间后,需要做某些修正、改变或扩展的情况下进行的维护测试。维护测试是在一个运行的系统上进行的,属于回归测试类型。

4.2.2　软件测试中的错误分级及其应用

软件测试的目的是暴露错误,评价程序的可靠性。而对软件错误进行级别定义或分级,目的就是科学地指导软件测试工作,提高软件测试的目的性,确保软件测试的质量。

1. 错误分级

软件错误分级涉及两个方面:错误分类及错误分级。不同的行业和企业、不同的应用领域以及不同的错误类型其错误的分级方法是不一样的。

1) 错误分类

软件错误分类有很多种方法,具体方法如下:

① 按软件生命周期分类有用户需求错误、产品需求错误、设计错误、编码错误、数据错误、发行错误。

② 按软件使用分类有功能错误、性能错误、界面错误、流程错误、数据错误、提示错误、常识错误以及其他错误。

③ 按 GB/T 15532—2008 分类有程序问题(软件不按照保障文档运行,但文档是正确的。

指程序编制过程中引入的错误)、文档问题(软件不按照保障文档运行,但其运行正确。指文档编写过程中引入的错误)、设计问题(软件因设计缺陷不按照保障文档运行。指软件设计过程中引入的错误)及其他问题(指不属于前面三种类型的错误)。

2) 错误级别划分

软件错误级别的划分同样有很多方法,GB/T 15532—2008 将对软件进行全面测试时所发现的错误分为如下级别。

第 1 级错误是有下列行为之一的软件问题:

① 妨碍由基线要求所规定的运行或任务的主要功能的完成。

② 妨碍操作员完成运行或任务的主要功能。

③ 危及人员安全。

第 2 级错误是有下列行为之一的软件问题:

① 给由基线要求所规定的运行或任务的主要功能的完成造成不利的影响,以致降低效能,且没有变通的解决办法。

② 给操作员完成由基线要求所规定的运行或任务的主要功能造成不利的影响,以致降低效能,且没有变通的解决办法。

第 3 级错误是有下列行为之一的软件问题:

① 给由基线要求所规定的运行或任务的主要功能的完成造成不利的影响,以致降低效能,但已知有变通的解决办法。

② 给操作员完成由基线要求所规定的运行或任务的主要功能造成不利的影响,以致降低效能,但已知有变通的解决办法。

第 4 级错误:这种软件问题给操作员带来不方便或麻烦,但不影响所要求的运行或任务的主要功能。

第 5 级错误:所有的其他错误。

2. 错误分级举例

下面是 GB/T 15532—2008 在某企业中的软件错误分级的实例化。

第 1 级:严重缺陷。即应用系统崩溃或系统资源使用严重不足:

① 系统停机(含软件、硬件)或非法退出,且无法通过重启恢复。

② 系统死循环。

③ 数据库发生死锁或程序原因导致数据库断连。

④ 系统关键性能不达标。

⑤ 数据通信错误或接口不通。

⑥ 错误操作导致程序中断。

第 2 级:较严重缺陷。即系统因软件严重缺陷导致下列问题:

① 重要交易无法正常使用,功能不符合用户需求。

② 重要计算错误。

③ 业务流程错误或不完整。

④ 使用某交易导致业务数据紊乱或丢失。

⑤ 业务数据保存不完整或无法保存到数据库。

⑥ 周边接口出现故障(需考虑接口时效/数量等综合情况)。

⑦ 服务程序频繁地要求重启(每天 2 次或以上)。

⑧ 批处理报错中断导致业务无法正常开展。

⑨ 前端未合理控制并发或连续点击动作，导致后台服务无法及时响应。
⑩ 在产品声明支持的不同平台下，出现部分重要交易无法使用或错误。
第 3 级：一般性缺陷。即系统因软件一般缺陷导致下列问题：
① 部分交易使用存在问题，不影响业务继续开展，但造成使用障碍。
② 初始化未满足客户要求或初始化错误。
③ 功能点得到实现，但结果错误。
④ 数据长度不一致，无数据有效性检查或检查不合理，数据来源不正确。
⑤ 显示/打印的内容或格式错误。
⑥ 删除操作不给提示。
⑦ 个别交易系统反应时间超出正常合理时间范围。
⑧ 日志记录信息不正确或应记录而未记录。
⑨ 在产品声明支持的不同平台下，出现部分一般交易无法进行或错误。
第 4 级：较小缺陷。即系统因软件操作不便方面缺陷：
① 系统某些查询、打印等实时性要求不高的辅助功能无法正常使用。
② 界面错误。
③ 菜单布局错误或不合理。
④ 焦点控制不合理或不全面。
⑤ 光标、滚动条定位错误。
⑥ 辅助说明描述不准确或不清楚。
⑦ 提示窗口描述不准确或不清楚。
⑧ 日志信息不够完整或不清晰，影响问题诊断或分析的。
第 5 级：其他缺陷。即系统辅助功能缺陷：
① 缺少产品使用、帮助文档，缺少系统安装或配置方面需要信息。
② 联机帮助、脱机手册与实际系统不匹配。
③ 系统版本说明不正确。
④ 长时间操作未给用户进度提示。
⑤ 提示说明未采用行业规范语言。
⑥ 显示格式不规范。
⑦ 界面不整齐。
⑧ 软件界面、菜单位置、工具条位置以及提示不美观，但不影响使用。

习题

1. 什么是软件配置项？什么是软件配置项测试，软件配置项测试有哪些测试内容？如何对它们进行分类并进行测试种类的选择？

2. 对单元测试、集成测试、配置项测试（也称软件合格性测试或确认测试）、系统测试、验收测试和回归测试进行名词解释。

3. 对功能测试、可靠性测试、性能测试、安全性测试、边界测试、安装性测试、余量测试、恢复性测试、接口测试、功能多余物测试和强度测试进行名词解释。

4. 什么是软件测试分级？简述软件生命周期测试分级及软件测试错误分级的思想及其应用。

第2部分

软件测试方法与技术基础篇

从第1部分的内容可以知道无论是传统的软件测试还是基于生命周期的软件测试，最终的目的就是发现软件中的各种错误，确保软件的质量，检验被测软件是否满足其规定的需求。尽管我们在软件测试生命周期中采用单元测试、集成测试、配置项测试（也称软件合格性测试或确认测试）、系统测试、验收测试和回归测试等测试类别，并对被测软件进行功能测试、可靠性测试、性能测试、安全性测试、边界测试、安装性测试、余量测试、恢复性测试、接口测试、功能多余物测试及强度测试等各种测试种类的测试，然而软件测试并不能保证发现和修复被测软件中所有的错误。因此怎样使得软件测试在资源和进度允许的条件下尽可能多地发现软件中存在的错误，从而使软件测试达到最佳的测试效果，即软件的质量得到更好的保证和提高，需求得到全面的满足，用户使用满意度高等，这就是我们要研究的内容，也是本部分要讲的内容：从软件测试方法（又可以称为软件测试技术）的角度来论述这个问题的解决答案。这些软件测试方法的科学使用将是软件测试任务高效率、高质量完成，软件质量得到充分保证，软件需求得到全面满足的技术保障。

软件测试是一项实践性很强的工作，它的发展是一个从实践到理论，又从理论回到实践不断往复的过程。而且随着软件开发技术、软件编程方法的发展，软件系统的不断扩大，结构越来越复杂，应用面越来越广，致使软件测试技术和测试方法也在持续发展。目前，软件测试方法主要分为静态测试和动态测试两大类，静态测试和动态测试下面又包含有很多测试方法或分为很多子类。如静态测试包括有代码审查、代码走查、桌面检查、技术评审和静态分析等；动态测试则包括"黑盒"测试、"白盒"测试以及"灰盒"测试等。

采用何种测试方法并如何使用该测试方法进行软件测试是测试阶段的关键技术问题。所谓测试方法包括具体的测试目的（例如，预定要测试的具体功能）、应该输入的测试数据和预期的结果。通常又把测试数据和预期的输出结果称为测试用例。其中最困难的问题是设计测试用例的输入数据。

不同的测试数据发现程序错误的能力差别很大，为了提高测试效率，降低测试成本，应选择高效的测试数据。因为不可能进行穷尽的测试，因此选用少量的"最有效的"测试数据，做到尽可能完备的测试就更重要了。

设计测试方案的基本目标是，确定一组最可能发现某个错误或某类错误的测试数据。已经研究出来的众多的测试方法，各有各的优缺点，不能说哪一种最好，哪一种最不好，更没有哪一种是可以替代其他的方法，完成测试；同一种测试方法在不同的环境下应用，得到的效果也是不一样的，因此通常都是多种测试方法联合使用，以达到最佳测试目的。

第5章 软件静态测试

静态测试通常是指不执行程序代码而寻找代码中可能存在的错误或评估程序代码的过程。其被测对象是各种与软件相关的有必要进行测试的产物，例如各类文档、源代码等。静态测试可以手工进行，也可以借助软件工具自动进行。静态测试具有以下特点：

(1) 静态测试不必动态地运行程序，也就是不必进行测试用例的设计和结果分析等工作。

(2) 静态测试可以人工进行，充分发挥人的思维的优势。在发现错误的同时也就可以定位错误。俗话说"解铃还需系铃人"，由于人的思维的局限性以及交流之间的障碍所造成的逻辑错误，由人通过逻辑思维去解决，是一种行之有效的方法，特别是在使得人的思维优势互补得到充分发挥后，测试的水平就会很高。

(3) 静态测试不需要特别的条件，容易展开。这是根据前两条得出来的。

(4) 静态测试对测试人员要求较高，至少测试人员要具有编程经验。

之所以要进行静态测试是因为一个软件可能暂时实现了需求说明书中的所有要求，但是由于它的内部结构复杂、混乱，代码的编写也没有规范，使得软件内部存在一些不易被察觉的错误，这些错误在特定的条件下会造成重大的影响；另外，软件虽然目前基本完成了用户的要求，但是随着时间的推移，软件产品需要升级维护，由于程序复杂或者代码编写杂乱，由此造成软件的维护工作很难进行。静态分析所要做的就是对代码标准以及质量进行监控，以此来提高代码的可靠性，使系统的设计符合模块化、结构化、面向对象的要求。静态分析主要是通过对源代码扫描或检查的方法来发现软件的缺陷，为日后的维护工作节省大量的人力、物力，从而更有效地保证软件的质量。

静态测试主要包括各阶段的评审、代码检查、程序分析、软件质量度量等。

5.1 各阶段评审

评审是对软件元素或项目状态进行评估的活动，用以确定它们与预期结果之间的偏差和相应的改进意见，通常由人来执行。除了在项目早期发现缺陷和降低项目失败风险外，项目中需要进行评审的其他原因包括分享知识、培训团队成员、为管理层决策提供依据、为过程改进提供信息以及项目所处状态评审。一般评审包括培训评审、预备评审、同行评审等，在这些评审中，我们最为关心的是同行评审。另外，需求阶段的规格说明书也是评审的重要内容。

5.1.1 同行评审

同行评审是由开发软件产品作者以外的其他人检查工作产品，以发现缺陷并寻找改进的机会。其评审方法是评审参与者通常采用逐行仔细阅读被评审对象的形式发现被测对象中的缺陷。评审的时间点一般设在里程碑点附近，即当工作产品到达了一个完成的里程碑并将进

入下一个开发阶段时，如图 5-1 所示。

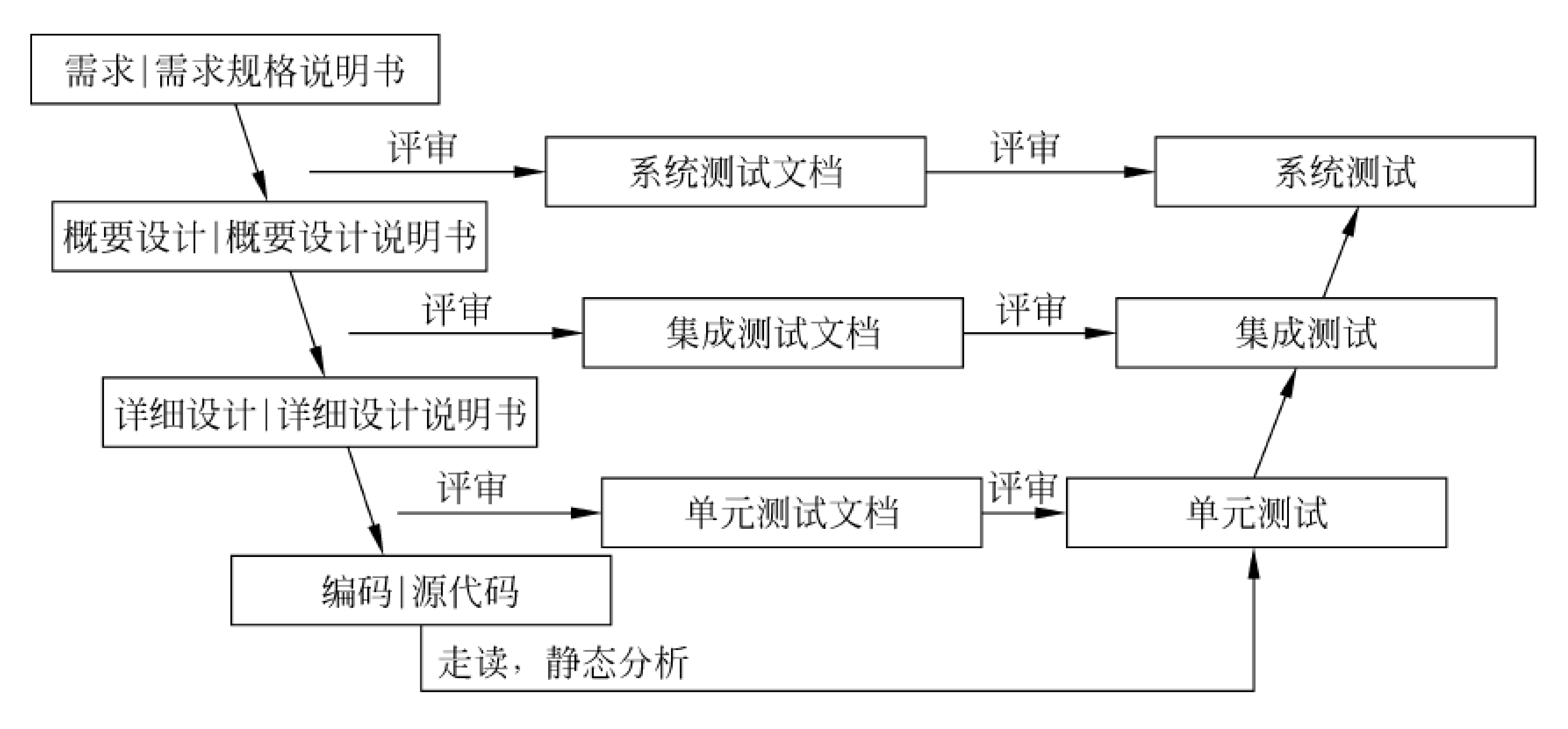


图 5-1 评审的时间点设置

同行评审一般包括审查、小组评审、走查、桌面评审、临时评审五种类型。

这些同行评审类型的区别在于正式程度：

① 审查是最正式，然后是小组评审、走查、桌面评审，临时评审最随意。

② 同行评审越正式，发现的缺陷越多，但评审越正式，花费成本越高。

③ 被评审对象越重要或者风险越高，采用的评审方式就越正式。

1. 审查

审查的概念是 IBM 的工程师 Michael Fagan 于 20 世纪 70 年代提出的，也叫正式评审，是一种包括非作者等专家在内的针对特定对象，如需求规格书、设计文档和源代码进行检查以发现缺陷的过程。

审查是一种有结构、有规则的评审方法。Fagan 的审查流程包括计划、介绍会议、准备、会议、返工、跟踪、因果分析。每个阶段需要确定的内容有参与审查的角色，相应的输入、输出等。图 5-2 所示为审查流程。

1）审查中的角色

（1）作者（被评审对象的创建者，提供被评审对象及其相关信息）。

（2）评审组长（组织评审会议，确保审查活动能够正确地进行）。

（3）审查专家（发现被评审对象中的问题）。

（4）读者（在会议上讲解被评审对象，使评审专家把精力集中在被评审对象本身而不是作者）。

（5）记录员（记录会议阶段有价值的信息）。

2）审查工作流程

（1）计划（参与者有作者和评审组长）。在这个阶段，需要开展这些工作：选择评审组长，确定审查对象，确定审查专家，确定总体会议、会议次数和相应的时间表，准备和分发审查工作包（审查包中包括被审查对象的初始可交付产品、相关参考文档、缺陷检查表、指导书、错误记录模板和其他材料）。

（2）总体会议。本阶段是可选的，主要目标是让审查专家熟悉被审查对象，包括对象特征、上下文、背景等。参与者可以是所有需要参加审查的人员。

（3）准备（参与者主要是审查专家，这是审查最重要的阶段）。在这个阶段，审查专家独立工作、逐行阅读被审查对象，将任何发现的问题、疑问记录在审查意见单中。评审组长根据各

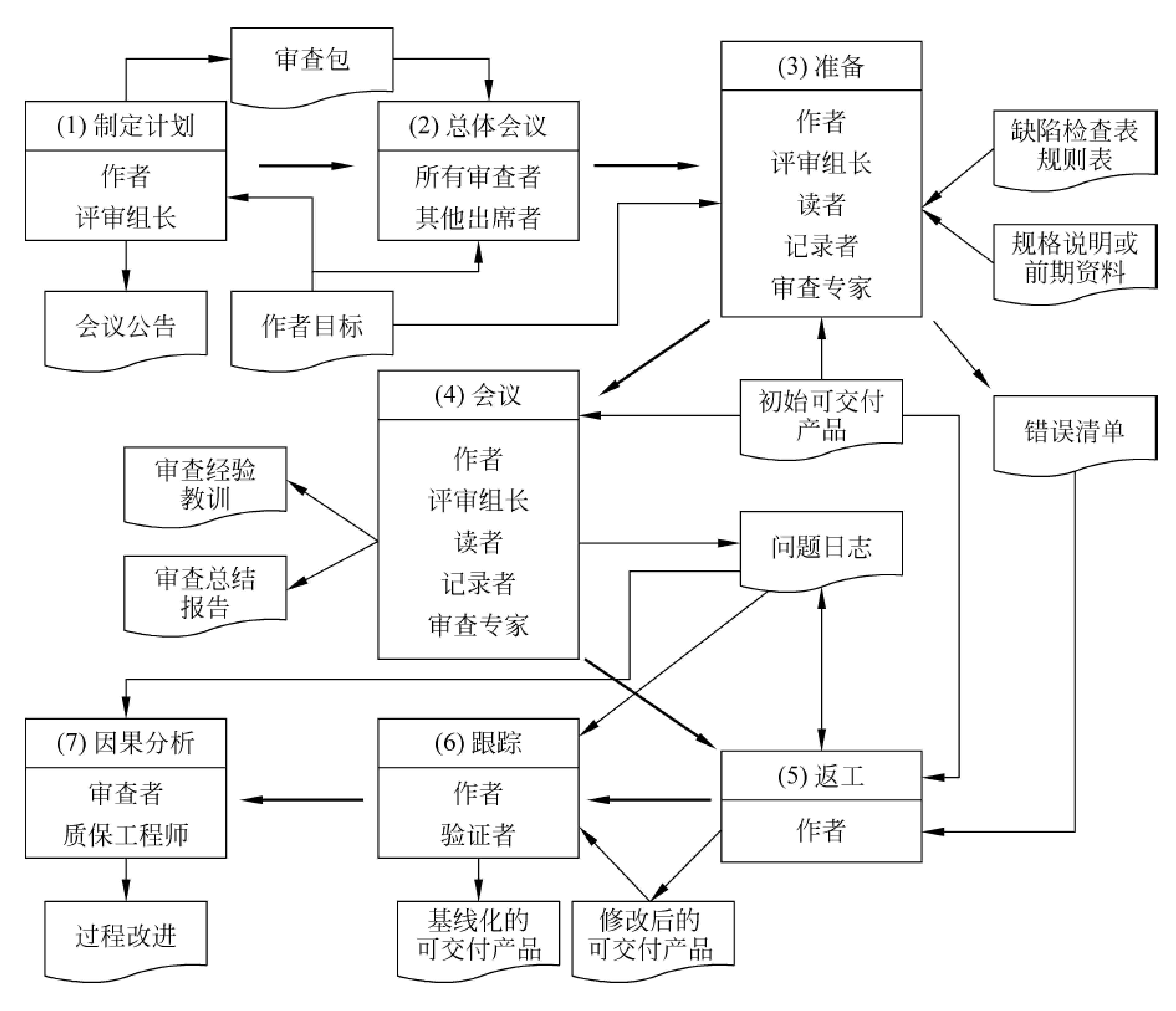


图 5-2 审查流程示意

个审查专家提交的意见决定是否按时或者推迟召开审查会议。

（4）会议(参与者有作者、评审组长、审查专家、读者、记录员)。在会议阶段,读者分段逐个阅读和检查被审查对象,审查专家听取讲解并考虑是否有新的问题提出。评审组长组织对所有审查意见单上的问题列表进行确认,作者确认是否问题,记录员在问题列表上记录答复和在会上发现的新缺陷。在会议结束前,所有人投票,给出对工作产品的审查结论。

（5）返工(参与者主要是作者)。在此阶段,作者修改会议中确认的问题,输出修改后的交付产品。

（6）跟踪(参与者有评审组长、质量工程师、指定的审查专家)。检查修改后的交付件,如果通过,则输出可进入基线的交付物。

（7）因果分析(参与者主要是质量工程师)。在这个阶段,开展分析缺陷原因、度量审查效率和效果的工作。

3）审查规则

为了更好地发挥审查的作用,在审查中有一组需要遵守的原则:

① 作者不能担当评审组长、读者或记录员等角色,要保持开放的思想,接受别人的意见,避免争论;

② 评审组长不要同时担任记录员;

③ 控制审查小组规模,3～7 个审查专家为好。

审查中需要遵守的原则有:

① 审查专家要努力发现被审查对象中的问题,审查过程中始终保持对问题的敏感性。

② 审查期间要努力发现问题,不要试图去解决问题。

③ 会议限制在两个小时之内。

④ 在会议上,审查团队要保持一个适当的审查速度,每小时 150～200 行代码或 3～4 页文档。

2. 小组评审

小组评审类似于审查,是一种“轻型审查”,可以采用前面审查中用到的指导方针和流程,只是没有审查正式,也没有审查严格,会议期间读者的角色由评审组长代替,小组评审方法发现问题的数量是审查的 2/3。

3. 走查和同级桌查

走查是产品的作者向同组同事说明该产品,希望获得他们的意见以满足自己的需要。走查是一种非正式的评审,其过程由作者主持,没有标准的流程可循。走查发现的缺陷数量比审查发现的缺陷数量要少一半。

同级桌查是一对一评审,是指除作者以外只有一位评审专家对工作产品进行检查。

4. 临时评审

临时评审是请团队内其他同事帮忙,在短时间内解决一些问题。

5. 软件评审指导书

软件评审指导书的用途是将评审过程和规则以指导书的形式固定下来,其内容包括目的、范围、评审角色及职责、过程准则、目标、进入标准、活动、退出标准、度量、相关资料、过程监控。

5.1.2 需求规格说明书的测试

检查软件的需求规格说明书一般采用逐行阅读说明书以发现缺陷的方式,需求规格说明书的测试应该在需求阶段规格说明书整体或者部分完成后立即开展。其目的是尽早地发现缺陷,使规格说明书具有更好的可测试性,软件测试人员可以更加熟悉系统应用。所采用的具体方法是静态黑盒测试(由于规格说明书非常重要,很多软件项目评审将规格说明书作为其重要的评审内容之一)。在进行规格说明书审查时可以采用对说明书进行概要评审和对说明书进行详细评审的技术。

1. 规格说明书的概要评审

对规格说明书进行概要评审所要达到的目的是发现特定的缺陷,比如大的原理性问题,遗漏或过度复杂的描述等。评审时,测试人员要站在用户的角度(确保作为第一质量要素的用户要求得到满足),研究现有标准和基线,对类似的软件系统进行评审和测试。

(1) 站在用户的角度问自己:我需要什么样的功能?我需要的所有功能是否都包含在规格书中了?是否存在与现有系统冲突的功能?功能是否易于使用?性能如何?功能的安全情况如何?等等。当然,测试人员如果能从一些熟悉软件目标应用领域的人那里获得支持,对评审过程将是非常有帮助的。

(2) 当对规格书进行概要评审的时候,测试人员应该参考现有的标准和基线,如:组织标准、术语和惯例(软件应该使用终端用户的通用术语和惯例),工业标准(在某些工业领域,例如通信、金融,有很多应用软件必须遵守的协议),政府标准,安全标准,等等。测试人员应该把相关标准作为规格说明书评审的一部分。

(3) 没有什么比经验更好的东西了,从类似软件中可以得到大量有用的信息,这些参考软件可能是:正在开发系统的早期版本,组织内的类似软件,竞争对手产品。分析类似软件的时候,应密切关注相关问题并考虑这些问题是否会影响被测试系统,如:特性是否有增删?代码

变更比例如何？软件的复杂度是否有区别？可测试性如何？性能、安全性和其他一些非功能特性如何？最后，不要忘了去从公共出版物和网上找有价值的信息。

2. 规格说明书的详细评审

测试人员对规格说明书进行详细评审，可从有关属性方面着手，一个好的规格说明书应具有如下属性。

（1）完整性：是否忘记或遗漏了什么内容，是否彻底，是否包含了该说明书所必须包含的所有信息。

（2）精确性：建议的提案是否正确，是否定义了合适的目标，是否有什么错误。

（3）准确性（明确而清晰）：描述是否清晰明确，是否有歧义，是否易于阅读和理解。

（4）一致性：特性描述内部和特性之间是否相互矛盾。

（5）相关性：细分特性是否必须，是否需要去除不必要的信息，特性是否可以跟踪到一个原始用户需求。

（6）可行性：项目计划和预算都是明确的，在给定的人力、工具和资源条件下，特性能否实现。

除了上述属性外，还可关注代码无关属性（规格书的目标是定义产品需求而不是软件设计、架构和代码）、可测试属性（特性是否可测试，是否提供了让测试人员得以验证功能的足够信息）。

检查规格说明书的同时，时刻关注评审的文字和图片是否具有这样的属性。

3. 问题词语列表

测试规格说明书的时候应密切关注下面的一些词汇以及这些词汇的上下文含义是否清晰。因为这些词汇常常会带来缺陷。

（1）总是、每个、所有、没有一个、从来不等。这些词表示肯定和确定的含义，必须确认用这些词语或不用这些词语的理由。

（2）当然、所以、明显地、无疑、显然等。这些词有劝说人接受的意思，规格书中尽量避免。

（3）一些、有时、经常、通常、大部分、主要的、等等、类似、好、快、便宜、高效、小和稳定等。这些词可测试性差，必须进一步定义以给出确切的含义描述。

（4）有把握的、处理过的、拒绝的、跳过的、去掉的等。这些词可能隐藏一些本该详细说明的功能性需求。

（5）如果……那么……等。这些描述依赖于其他因素，不可取。

5.2 代码检查

代码检查是“白盒”测试的一种静态测试方法，是众多软件测试方法中发现软件缺陷最有效的方法之一。主要是由检验人员通过仔细地分析代码，检查代码和设计的一致性、代码对标准的遵循、代码的可读性、代码的逻辑表达正确性、代码结构的合理性等。例如，一些问题是代码虽然可以正常运行，但是代码的编写不符合某种标准或规范，这就相当于是写出来的东西可以被人们理解和使用，但是不符合语言的语法和文法规范。标准是建立起来的、经过修改和必须遵守的规则——做什么和不做什么，规范是建议最佳做法，标准是必须遵守的，规范可以适当地放宽。

有的代码可以运行，甚至在测试中也表现出了稳定性，但是因为它不符合某些标准而仍被认为是有问题的代码，造成这种现象的原因有很多，但主要是三个重要原因：

(1) 可靠性。大量的事实证明按照某种标准和规范编写的代码比不这样做的代码更加可靠和安全。

(2) 可读性和可维护性。符合某种标准和规范的代码易于阅读、理解和维护。

(3) 移植性。代码经常要在不同的平台上运行或者使用不同的编译器进行编译。如果代码符合设备标准,迁移到另一个平台就会轻而易举,甚至完全没有问题;相反,程序在编写时未考虑移植的问题,测试时也只在一个平台上进行,那么当需要迁移到另一个平台时就要做大量的修改甚至是重新编写。

在代码全部或部分完成后,应立即进行逐行代码评审以发现缺陷。以达到尽早发现缺陷,使得程序更具可测试性,软件测试人员可以更加熟悉系统的目的。

代码检查要求评审人员有程序开发语言的专业知识,有程序基线和标准供参考。

代码检查的内容有:

(1) 完整性检查(代码是否完全实现了设计文档中提出的功能需求,代码中是否存在任何没有定义或没有引用到的变量、常数或数据类型)。

(2) 一致性检查(代码的逻辑是否符合设计文档,代码中使用的格式、符号、结构等风格是否保持一致)。

(3) 正确性检查(代码是否符合制定的标准,所有的变量是否都被正确定义和使用,所有的注释是否都是准确的)。

(4) 可修改性检查(代码涉及的常量是否易于修改。如使用配置、定义为类常量、使用专门的常量类等)。

(5) 可预测性检查(代码是否具有定义良好的语法和语义,代码是否无意中陷入了死循环,代码是否避免了无穷递归)。

(6) 健壮性检查(代码是否采取措施避免运行时错误。如空指针异常等)。

(7) 可理解性检查(注释是否足够清晰地描述每个子程序,对于没用的代码注释是否删除;是否使用到不明确或不必要的复杂代码,它们是否被清楚地注释;使用一些统一的格式化技巧用来增强代码的清晰度,诸如缩进、空白等;是否在定义命名规则时采用了便于记忆、反映类型等方法;循环嵌套是否太长太深)。

(8) 可验证性检查(代码中的实现技术是否便于测试)。

(9) 结构性检查(程序的每个功能是否都作为一个可辨识的代码块存在,循环是否只有一个入口)。

(10) 可追溯性检查(代码是否对每个程序进行了唯一标识,是否有一个交叉引用的框架可以用来在代码和开发文档之间相互对应,代码是否包括一个修订历史记录,记录中对代码的修改和原因都有记录,是否所有的安全功能都有标识)。

5.2.1 代码检查方法

所谓的代码检查,其实就是以组为单位阅读代码,是一系列规程和错误检查技术的集合。该过程通常将注意力集中在发现错误上,而不是纠正错误。

代码检查一般采用静态"白盒"测试的方法,如代码审查、桌面检查、代码走查和技术评审。其中代码走查和代码审查是由若干个程序员和测试人员组成的一个小组,集体讨论。这两个方法都需要先做一些准备工作,然后才举行会议进行讨论,会议的主题是找出软件的问题——不仅是出错的项目,还包括遗漏的项目,但不解决问题。很多软件项目团队选择审查作为评审核心代码的方式,采用走查和同级桌查作为一般代码的评审方式。

1. 代码审查

代码审查的内容有代码和设计的一致性、代码执行标准的情况、代码的逻辑表达正确性、代码结构的合理性、代码的可读性等。代码审查是在开发组内部进行的，类似于“如果你给我看你的，我也给你看我的”。

代码审查一般不少于 4 人，分别为组长、资深程序员、程序编写者与专职测试人员。组长不能是被测程序的编写者，组长负责分配资料、安排计划、主持开会、记录并保存被发现的问题。

代码审查组采用讲解、提问并使用检查表的方式审查代码，寻找问题和失误。一般有正式的计划、流程和结果报告。同时为了保证审查的效率，所有的参与者要保证正式审查的 4 个关键要素：查找问题、遵守规则、审查准备和编写报告。

代码审查由 4 个步骤组成：准备、程序阅读、审查会、编写报告。

(1) 准备：将程序目录表和设计说明书等材料交给小组成员，要求大家熟悉这些材料，并由设计人员和编程人员对所提交的材料进行说明，以了解代码的主要功能和各个功能之间的联系。

(2) 程序阅读：审查组人员在对程序有了一定的了解后，仔细阅读代码和相关的材料，标出明显的缺陷及错误。

(3) 审查会：由程序员讲解程序的逻辑，其他人员提出问题，确定错误是否存在，然后采用代码审查会来分析讨论，切记是发现问题，不是解决问题。

(4) 编写报告：在会后审查人员除了要编写审查结果外，还要将发现的错误编写成报告交给程序开发人员，其中错误报告也要分析、归类和精炼。审查会议的结果必须尽快提交——例如发现了多少错误，在哪里发现的。

代码审查过程中最主要的是代码审查清单，下面给出一种常用的代码审查清单。

(1) 数据引用错误。主要有：

① 是否引用未初始化的变量？查找遗漏之处和查找错误同样重要。

② 数组和字符串的下标是整数值吗？下标是否总是在数组和字符串的长度范围之内？

③ 用下标引用数组时是否存在“差 1”错误？

④ 在引用指针或变量时是否已分配内存？

(2) 数据声明错误。主要有：

① 所有变量是否都赋予正确的长度、类型和存储类？

② 是否存在已声明但从未用过的变量？

③ 所有变量是否都显式地声明了？

(3) 计算错误。主要有：

① 计算中是否使用了不同数据类型的变量？

② 计算中是否存在混合运算？

③ 计算中是否出现了溢出现象？

④ 对于整型运算，处理某些计算是否存在精度丢失问题？

⑤ 除数/模是否为零？

⑥ 赋值语句的目标变量是否比其有变量的表达式的取值范围要小？

⑦ 是否考虑和了解到编译器对类型和长度不一致的变量的转换规则？等等。

(4) 子程序参数错误。主要有：

① 子程序接受的参数类型与调用代码发送的匹配吗？

② 常数是否当做形参传递，在子程序中被改动？

③ 参数类型是否与相应的形参匹配？

④ 是否存在子程序中定义了与全局变量相同的变量？

(5) 静态结构问题。主要有：

① 函数的调用关系是否正确。

② 是否存在孤立的函数而没有被调用。

③ 编码的规范性。

④ 资源是否释放。

⑤ 数据结构是否完整和正确。

⑥ 是否有死代码和死循环。

⑦ 代码本身是否存在明显的效率和性能问题。

⑧ 代码本身方法、类和函数的划分是否清晰并易理解。

⑨ 代码本身是否健壮，是否有完善的异常处理或错误处理功能。

其他的问题还有控制流错误、比较错误、输入输出错误等。

2. 桌面检查

桌面检查就是程序员自己检查自己所编写程序（根据相关的文档，对源程序代码进行分析、检验，查找程序中是否存在错误）的过程。

桌面检查主要做的工作是：

① 检查代码和设计是否一致。

② 代码是否遵循标准、是否可读。

③ 代码逻辑表达是否正确。

④ 代码结构是否合理。

⑤ 程序编写与编写标准是否符合。

⑥ 程序中是否有不安全、不明确和模糊的部分。

⑦ 编程风格是否符合要求。

桌面检查所要求的更加细致的工作是：

① 检查变量的交叉引用表（是否有未说明的变量和违反了类型规定的变量）。

② 检查标号的交叉引用表（验证所有标号是否有正确）。

③ 检查子程序、宏、函数（验证每次调用与所调用位置是否正确，调用的子程序、宏、函数是否存在，参数是否一致）。

④ 检查全部等价变量的类型的一致性。

⑤ 确认常量的取值和数制、数据类型。

⑥ 选择和激活路径（在设计控制流图中选择某条路径，到实际的程序中激活这条路径，如果不能激活，则程序可能有错）。

⑦ 对照程序的规格说明，详细阅读源代码，比较实际的代码，从差异中发现程序的问题和错误。

桌面检查是一种很老的方法，但是它的效率不高，原因是：

① 人都有思维定势，有些习惯性错误自己很难发现。

② 人们对于自己的程序都有一种偏爱心理，没有发现错误的欲望。

③ 如果是对功能的理解错误，则自己很难纠正。因此这种方法只能用做个人自我检查和发现一些较明显的错误和漏洞。

3. 代码走查

代码走查和代码审查类似,代码审查是一种正式的评审活动,而代码走查的讨论过程是非正式的。当然走查比审查要更技术性些。在代码走查中编写代码的程序员要向5人小组或者其他程序员和测试人员组成的小组做正式陈述。在这个小组中至少有一位资深程序员,测试人员应是具有经验的程序设计人员,或精通程序设计人员,还要有一位没有介入到这个项目中的新人(这样的人不会被已有的设计约束)。

代码走查的过程和代码审查相似,先把材料交给审查者,审查者阅读材料发现错误,在审查会期间由陈述者(编写代码的程序员)逐行或逐段地通读代码,解释代码为什么这样工作。审查人员聆听叙述、提出有疑义的问题,最后审查小组做出审查结果的书面报告和错误报告,交给程序开发人员。

代码走查主要针对文档和源程序代码,通常检查功能、检查界面、检查流程、检查提示信息以及其他的检查(如函数检查、数据类型与变量检查、条件判断检查、循环检查、输入输出检查、注释检查、程序/模块检查、数据库检查、表达式分析、接口分析、函数调用关系图及模块控制流图等),检查内容共计17项。

(1) 要求有文档和源程序代码(一份最新的设计文档、程序结构图、所有模块的源程序代码、代码体系结构描述、目录文件、代码组织等)。

(2) 检查功能(重复的功能,多余的功能,功能实现与设计要求不相符,功能的可使用性、方便性和可用性)。

(3) 检查界面(界面美观,控件排列、格式,焦点控制合理或全面性)。

(4) 检查流程(流程控制是否符合要求,流程实现是否完整)。

(5) 检查提示信息(提示信息重复或出现时机的合理性,提示信息格式和要求的合理性,提示框返回后停留位置的合理性)。

(6) 函数检查(函数声明部分清楚地描述函数及其功能,代码中有相关注解,函数名清晰地定义了它的目标以及函数的功能,函数只做一件事情,函数的参数都被使用,函数的参数接口关系清晰,函数出口都有返回值,函数异常处理清楚)。

(7) 数据类型与变量检查(数据有效性检测是否合理,数据来源正确性,数据处理过程正确性,数据处理结果正确性,数据类型解释,变量分配了正确的长度、类型和存储空间,静态变量明确区分,变量初始化,变量的命名不与标准库中的命名相冲突,全局变量描述,类型转换)。

(8) 条件判断检查(if/else 使用正确,无嵌套的 if 链,数字、字符、指针和 0/NULL/FALSE 判断明确,不要有臃肿的判断逻辑,所有的判断条件边界是否正确,判断体足够短)。

(9) 循环检查(循环体不为空,循环之前做好初始化代码,有明确的多次循环操作可使用 for 循环,不明确的多次循环操作可使用 while 循环,循环终止的条件清晰,所有的循环边界是否正确,循环体内的循环变量起到指示作用)。

(10) 输入输出检查(所有文件的属性描述清楚,输入参数的异常是否处理了,对文件结束的条件进行检查)。

(11) 注释检查(有一个简单的说明用于描述代码的结构,每个文件和模块均已给予解释,解释说明代码功能且准确描述代码意义,解释不要过于简单,注解清楚正确,代码的注释与代码是否一致且注释是否多余)。

(12) 程序(模块)检查(程序中所有的异常是否处理了;程序中是否存在重复的代码,程序结构是否清晰)。

(13) 数据库检查(数据库命名使用小写英语字母、数字和下划线,数据库命名采用项目名

称或产品名称命名，数据库中的所有表字符集统一，数据库对象的命名不使用保留关键字，数据库设计考虑到将来可能存在的异种数据库迁移，索引是多值字段，索引是单一字段，字段取值符合域定义，字段的类型和长度能够满足字段的值的最大限量，文本字段有充足的余量对应可能的长度变更，数字字段考虑了充足的余量和精度以对应可能的长度或精度变更，针对客户的特定应用采用了视图机制）。

（14）表达式分析（对表达式进行分析以发现和纠正在表达式中出现的错误。如在表达式中不正确地使用了括号造成错误、数组下标越界错误、除数为零、浮点数计算的误差等）。

（15）接口分析（主要是对接口一致性的分析。如各模块之间接口一致性，模块与外部数据库的接口一致性，形参与实参在类型数量、顺序、维数、使用上的一致性，全局变量和公共数据区在使用上的一致性）。

（16）函数调用关系图（通过应用程序各函数之间的调用关系展示系统的结构。方法是列出所有函数，用连线表示调用关系。其作用是可以检查函数的调用关系是否正确，是否存在孤立的函数而没有被调用，明确函数被调用的频繁度，对调用频繁的函数可以重点检查）。

（17）模块控制流图（模块控制流图是由许多节点和连接节点的边组成的图形，其中每个节点代表一条或多条语句，边表示控制流向，可以直观地反映出一个模块的内部结构）。

4. 技术评审

技术评审是最正式的审查类型，具有高度的组织化，要求每一个参与者都接受训练。技术评审由开发组、测试组和相关人员（QA、产品经理等）联合进行，综合运用走查、审查技术，逐行、逐段地检查软件。技术评审与走查和审查的不同之处在于表述代码的人——表述者，表述者不是原来编写代码的程序员，这就迫使表述者学习和了解要表述的材料，从而有可能对程序提出不同的看法和解释。检查的要点是设计需求、代码标准/规范/风格和文档的完整性与一致性。

经验表明，通过代码审查、桌面检查、代码走查和技术评审能够有效地发现30％～70％的逻辑设计和编码错误，而且这种方法一次能处理一批错误，同时还能对错误进行定位。

5.2.2 代码编程规范检查

编程规范又称为代码规则、编码规则，是对程序代码的格式、注释、标识符命名、语句使用、函数、类、程序组织、公共变量等方面所做的要求。如果软件都能在编写的阶段遵循一定的编程规范，这对软件产品的质量将大有好处：遵循一定的编程规范，使得开发人员书写的代码更健壮、更安全、更可靠，可以提高代码的可读性，使代码易于查看和维护，是提高代码质量最有效、最直接的手段。

所谓编程规范即是千百万有经验的程序员经历长期教训后，由少数的一些权威人士和专家通过总结和反思而养成的信条和习惯，是一种能够极大地提高代码的可读性、可重用性、程序健壮性、可移植性和可维护性等性能的有效机制。不少公司大力推行CMM（软件能力成熟度模型）或者TSP（团队软件过程），也有不少公司对程序员进行PSP（个人软件过程）的推行，无论推行什么开发过程，目的都只有一个，就是要求规范统一，以便整个开发团队便于交流、步调一致。遵守编程规范是一名合格的程序员的必要条件，也是从业余程序员蜕变为职业程序员的重要标志。在公司团队协作开发的情况下，编程时应该强调的一个重要方面是程序的易读性，在保证软件的速度等性能指标满足用户需求的情况下，能让其他程序员容易读懂你的程序。进一步而言，简明的编程风格，可以让协作者、后继者和自己一目了然，在很短的时间内看清程序的结构，理解设计的思路，从而可极大而有效地促进程序员之间的交流。

然而对于各个不同的领域、行业、语言、操作系统、版本、公司、团队甚至不同的项目，都需要符合自己特点的编程规范，编程规范的种类可以说是五花八门、数不胜数。虽然有一些由业内比较著名的组织或公司等指定的相对权威的编程规范，但由于矛盾的特殊性，这些权威的编程规范更多的是被参考和引用，而并不能被应用在所有的应用领域。比如说编译程序和解释性程序的编程规范是明显不同的，基于 Web 的语言更有它独特的要求，有时如果强制性地遵循某些权威或者比较流行的规范，可能会导致效率下降或可读性不强等问题。

下面具体讲述一下编码规则的具体内容。

1. 对代码书写格式的要求

下面比较两段代码：

程序 A：

```
int main()
{
  int i,b,c,a[] = {...}, * p, * q;
  b = c = 1;p = q = a;
  for(i = 0;i < 6;i++)
  {if(b < * (a + i)){b = * (a + i);p = &a[i];}if(c > * (a + i)){c = * (a + i)q = &a[i]j;}}
  i = * a; * a = * p; * p = i;i = * (a + 5); * (a + 5) = * q; * q = i;
  printf(" % d, % d, % d, % d, % d, % d\n",a[0],a[1],a[2],a[3],a[4],a[5]);
}
```

程序 B：

```
int main()
{
    int i,b,c,a[] = {...}, * p, * q;

    b = c = 1;
    p = q = a;

    for(i = 0;i < 6;i++)
    {
        if(b < * (a + i))
        {
            b = * (a + i);
            p = &a[i];
        }

        if(c > * (a + i))
        {
            c = * (a + i)
            q = &a[i];
        }
    }

    i = * a;
    * a = * p;
    * p = i;

    i = * (a + 5);
    * (a + 5) = * q;
    * q = i;
```

```
    printf("%d,%d,%d,%d,%d,%d\n",a[0],a[1],a[2],a[3],a[4],a[5]);
}
```

实现同样功能的两段程序，都可以编译通过，正确运行。但是两者相比较而言，程序 B 明显比程序 A 的可读性好，程序 B 可以很清楚地看到程序的模块，查找错误时也非常方便，而程序 A 虽然可以编译通过，正确运行，但是如果想做什么修改或维护，则很难清楚明白整个程序，无从下手；另外，程序 B 比程序 A 占的空间大，但是在程序编译运行中，空格和空行是没有影响的。下面详细介绍代码书写的格式。

1）空行、空格的使用

由于在程序的编译和运行过程中，空行和空格没有影响，因此可以利用空行和空格使程序看起来整齐、清楚。如同上面的程序 B 一样，利用空行和空格将程序在哪一步做了什么事情划分得清清楚楚，使人一目了然。

2）对程序语句要按其逻辑水平缩进

缩进可以让人清楚地看到程序的逻辑结构，有助于对程序进行维护。例如在程序 B 中，可以清楚地看到 for 循环语句作用范围，也可以清楚地检查程序中是否存在逻辑错误。尤其是在很复杂的程序中，存在很多的嵌套循环，可以利用缩进清楚明白地看到程序块的逻辑结构。

3）一行只写一条语句，一次只声明、定义一个变量

在程序 A 中一行写了很多条语句，使人看起来眼花缭乱，在查找错误时，不易查找；程序 B 中，每一行只写一条语句，再加上空行和空格的使用，使程序看起来简洁清楚，易于查错和维护。

4）在表达式中使用括号

表达式 1：a>b&&x>y

表达式 2：(a>b)&&(x>y)

表达式 1 和表达式 2 表示的是同样的意思，但是表达式 2 看起来就很清楚，不会让人产生歧义，而表达式 1 则会产生歧义。由于运算符有优先级，不同的运算符的优先级不同，运行的先后自然也不一样，但是当表达式中有众多运算符时，最好加上括号以表明运算符运行的优先级，这样方便查找程序中存在的逻辑错误，利于维护。

2. 程序的注释

程序中的注释是程序与日后程序读者之间通信的重要手段，良好的注释能够帮助读者理解程序，为后续阶段进行测试和维护提供明确的指导。

注释的基本原则：

① 注释内容要清晰明了，含义准确，防止出现二义性；

② 边写代码边写注释，修改代码的同时也要修改相应的注释，以保证代码与注释的一致性。

通常需要添加注释的有函数、类、文件、空循环体、swich 语句中多个 case 语句共用一个出口及其他，在行末注释时尽量对齐注释。在程序中注释行的数量不得少于程序行数量的 1/3。

3. 命名

不同的编程语言对于变量、文件名、常量、函数的命名都有各自统一的命名规范。例如在 C 语言中，用来标识变量名、符号常量名、函数名、数组名、类型名、文件名的有效字符序列统称为标识符，标识符只能由字母、数字和下划线组成，而第一个字符必须为字母或下划线。例如，下面列出的就是合法的标识符：

```
Sum, li_ling, class1, _day3
```

下面的是不合法的标识符：

M.D.， $123， 34th， a>b

另外，在C语言中还要求标识符的长度不能超过32个字符，还要尽量做到“见名知意”，即选有含义的英文单词或缩写作标识符，如count，name，day，month等，符号常量名用大写，变量名用小写。文件名的命名类似。

4. 语句

对于具体程序语句的使用要求，不同的编程语言有不同的要求，在这里简单介绍几点。

(1) 禁用goto语句，goto语句使程序的流程无规律、可读性差。

(2) new和delete要成对出现，new用来分配一块新的内存，delete用来释放一块内存，如果两者不成对使用，将会造成内存的浪费。

(3) 对switch语句中每个分支都要以break语句结尾。

(4) 指针要初始化，释放内存后的指针变量要赋NULL值。

5. 函数

在函数声明和定义时，要在函数参数列表中为各个参数指定类型和名称；为每一个函数指定其返回值，若没有返回值，则要定义返回类型为void；若函数体代码的长度超过100行(不包括注释行)，则需要重新编写函数；另外在函数中要注意全局变量的使用。

6. 类

类成员变量的声明必须写在该模块中所有可执行代码之前；应当至少声明一个构造函数，类的构造函数应当出现在所有成员函数的最前面，而且应当以参数个数递增的顺序排列；类成员变量和成员函数的出现应当根据其可访问性的级别来定；在派生类中不要对基类中的非虚函数重新进行定义，如果确实需要在派生类中对该函数进行不同的定义，那么应在基类中将该函数声明为虚函数；用内联函数代替宏函数；若重载了操作符new，则也要重载delete；虽然类可以继承，但是仍要限制类的继承层数(最好不要超过5层)。

7. 程序组织

由于程序的规模越来越大，软件往往包含多个文件。因此，在程序的组织中要注意：

(1) 一个头文件中只声明一个类。

(2) 一个源文件中只实现一个类。

(3) 头文件中只包含声明，不应包含定义或实现。

(4) 源文件中不要有类的声明。

(5) 只允许头文件被包含到其他代码文件中。

(6) 避免头文件的重复包含。

有了统一的规范后，测试工程师或者程序员自身，就可以实施编码规范检查了。要真正把编码规范贯彻下去，单单靠测试员、程序员的热情，很难坚持下去，我们必须借助于一些专业的工具来实施。在C/C++语言的编程规则检查方面，比较专业的工具有C++Test、LINT、QAC/QAC++等，这些工具通常可以和比较流行的开发工具集成在一起，程序员在编码过程中，在编译代码的同时即完成了编程规则的检查。

5.2.3 代码的自动分析

代码的自动分析需要用到代码分析工具，代码自动分析的结果可以对照着需求和设计文档以及编码进行检查，主要进行程序逻辑和编码检查，一致性检查，接口分析，I/O规格说明分析，数据流、变量类型检查和模块分析等。代码自动分析的结果可以作为动态测试和其他测试

的必要准备。

运用代码分析工具进行代码自动分析的内容主要是生成引用表、进行程序错误分析和接口分析。

1. 生成引用表

代码分析所生成的引用表有循环层次表、变量交叉引用表、标号交叉引用表、子程序引用表、等价表、常数表、操作符统计表和操作数统计表。

生成引用表的目的：直接从表中查出说明、使用错误，如循环层次表、变量交叉引用表、标号交叉引用表；为用户提供辅助信息，如子程序引用表、等价表、常数表；用来做错误预测和程序复杂度的计算，如操作符和操作数的统计表。

1）标号交叉引用表

列出各模块出现的全部标号（其顺序可以是按标号出现的先后顺序，也可以按字典顺序）；在表中标出标号的属性：已说明、未说明、已使用、未使用；在表中还可以包括模块外的全局标号、计算标号。

2）变量交叉引用表

变量交叉引用表也称变量定义与引用表（在表中，变量的顺序可以是按变量出现的先后顺序，也可以按字典顺序，还可以按它们的类型排序）；表中应标明各个变量的属性：已说明、未说明、私有/公有说明，以及类型和使用情况。

3）子程序、宏和函数表

在表中列出各个子程序、宏和函数的属性，包括已定义、未定义、定义类型；已引用、未应用、引用次数；输入参数的个数、类型、顺序；输出参数的个数、类型、顺序。

4）等价表

在表中列出在等价语句或等值语句中出现的全部变量和标号。

5）常数表

在表中列出全部的数字常数和字符常数，并指出它们在哪些语句中首先被定义。

2. 程序错误分析

程序错误分析目的是用于确定在源程序中是否有某类错误或危险结构。分析的内容有变量类型和单位分析、引用分析以及表达式分析。

1）变量类型和单位分析

变量类型和单位分析即为了强化对源程序中数据的检查，在程序设计语言中扩充一些新的数据类型，例如，仅能在数组中使用的“下标”类型及在循环语句中当做循环变量使用的“计数器”类型。这样就可以应用静态预处理程序，分析程序中的类型错误。

2）引用分析

在静态错误分析中，最广泛使用的技术就是发现引用异常。例如，沿着程序的控制路径，变量在赋值以后未被引用，这就发生了引用异常。为此，我们需要检查通过程序的每一条路径。可以用类似深度优先算法的方法来遍历程序流程中的每一条路径；建立引用异常的探测工具。这种工具包括两个表：定义表和引用表。每张表中都包含一组变量名。未引用的表中包括已被赋值，但未被引用的一些变量。

3）表达式分析

对表达式进行分析，可以发现和纠正在表达式中出现的错误。在表达式中可能存在：不正确使用括号造成的错误，数组下标越界造成的错误，除数为0造成的错误，对负数开平方造成的错误。其中最复杂的一类表达式分析是对浮点数计算的误差进行检查。

3. 接口一致性分析

接口一致性分析的目的是检查模块之间接口的一致性和模块与外部数据库之间接口的一致性。其分析内容主要有：

（1）检查形参与实参在类型、数量、维数、顺序、使用上的一致性。

（2）检查全局变量和公共数据区在使用上的一致性。

5.2.4　代码结构分析

代码的结构形式是“白盒”测试的主要依据。研究表明程序员38%的时间花费在理解软件系统上，因为代码以文本格式被写入多重文件中，这是很难阅读理解的，需要其他一些东西来帮助人们阅读理解，如各种图表等，而代码结构分析满足了这样的需求。

在代码结构分析中，测试者通过使用测试工具分析程序源代码的系统结构、数据结构、内部控制逻辑等内部结构，生成函数调用图、模块控制流图、模块数据流图、内部文件调用关系图、子程序表、宏和函数参数表等各类图形、图表，可以清晰地标识整个软件系统的组成结构，便于阅读和理解，然后可以通过分析这些图表，检查软件有没有存在缺陷或错误。

1. 函数调用关系图

函数调用关系图/程序调用关系图都是对源程序中函数关系的一种静态描述，即通过应用程序中各函数之间的调用关系展示系统的结构。在函数调用关系图中，节点表示函数，边表示函数之间的调用关系。

通过查看函数调用关系图，可以检查函数之间的调用关系是否符合要求，是否存在递归调用，函数的调用是否过深，是否存在独立的没有被调用的函数。从而可以发现系统是否存在结构缺陷，发现哪些函数是重要的，哪些是次要的，需要使用什么级别的覆盖要求，等等，如图5-3所示。

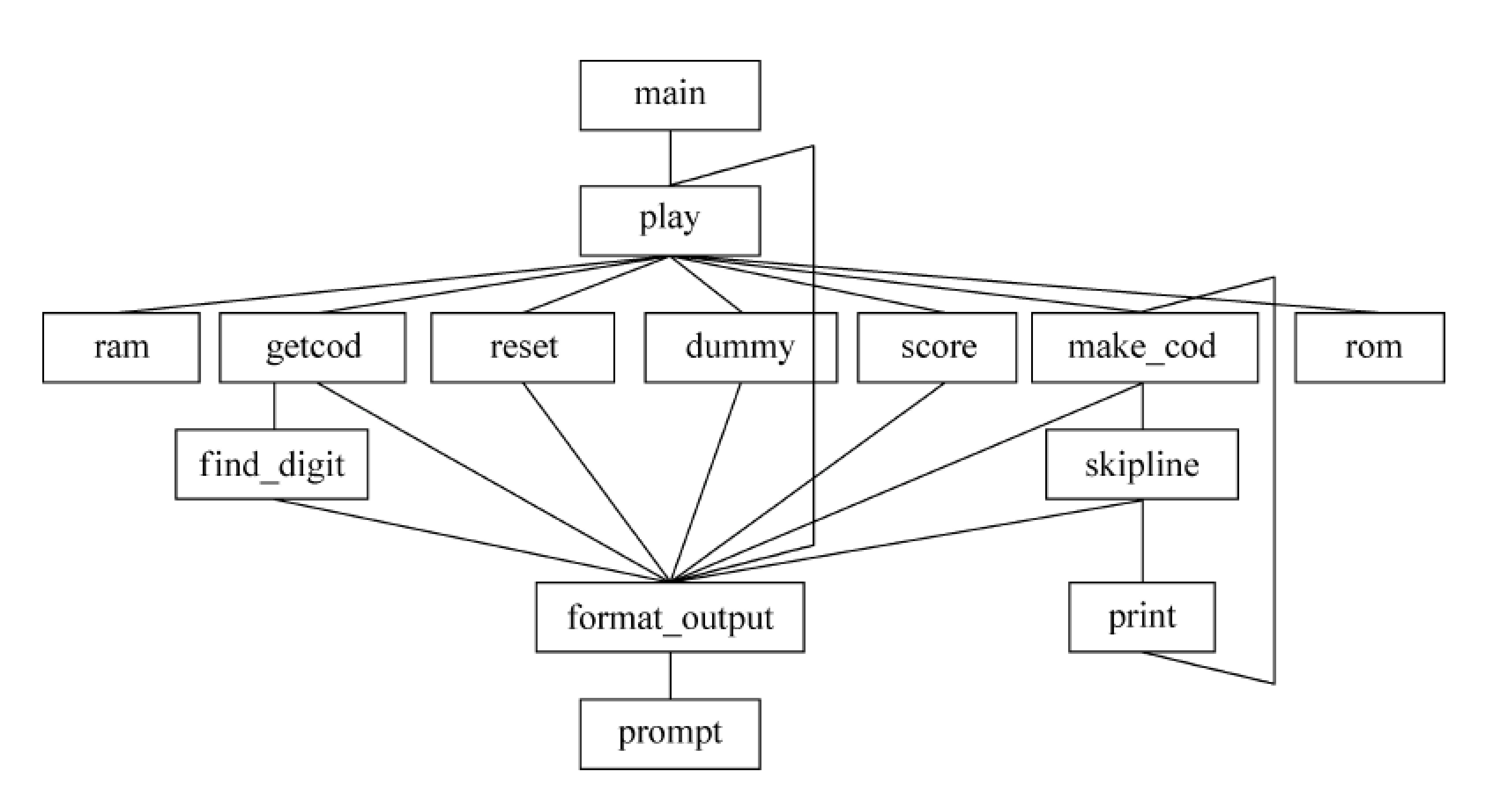


图5-3　函数调用关系图

图5-3指出了程序中的很多问题：程序的层次性较差、存在直接和间接的递归调用以及重要资源被诸多模块使用等。

函数调用关系图在软件工程领域有广泛的应用，如编译优化、过程间数据流分析、回归测试、程序理解等。

2. 模块控制流图

模块控制流图是与程序流程图相类似的由许多节点和连接节点的边组成的一种图形，其

中一个节点代表一条语句或数条语句，边代表节点间控制流向，它显示了一个函数的内部逻辑结构。模块控制流图可以直观地反映出一个模块或函数的内部逻辑结构，通过检查这些模块控制流图，能够很快发现软件的错误与缺陷。

通过控制流图，可以很直观地发现程序中的问题，例如，在图 5-4 中，我们发现程序中使用了 GOTO 语句、代码重复、开关语句结构有问题以及死代码等。

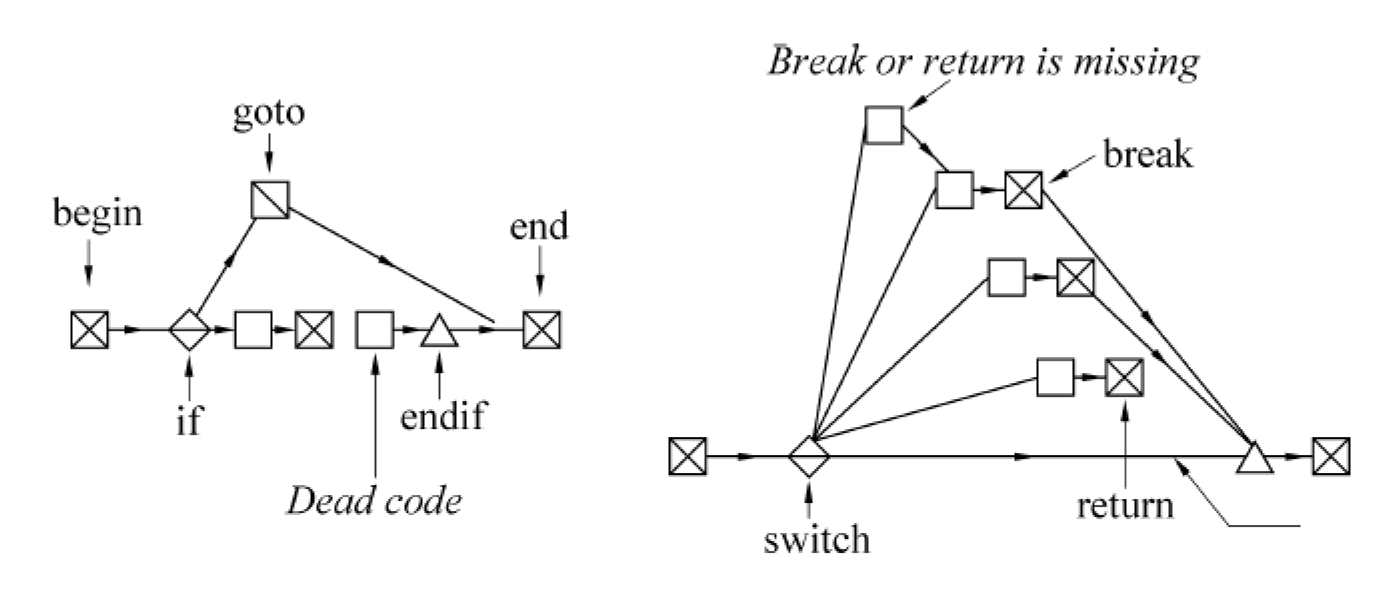


图 5-4 模块控制流图

因此，对于程序结构提出 4 点基本要求，这些要求是，编写的程序不应该包含：

① 转向并不存在的标号。

② 没有用过的语句标号。

③ 从程序入口进入后无法达到的语句。

④ 不能达到停机语句的语句。

3. 模块数据流图

在单元测试中，数据仅仅在一个模块或者一个函数中流通。但是，数据流的通路往往涉及多个集成模块，甚至于整个软件，所以我们有必要进行数据流的分析，尽管它非常耗时。

数据流分析技术最早被用于编译优化，目前除编译优化以外，在程序测试、程序理解、程序验证、程序调试以及程序分片等许多领域，数据流都有着广泛的应用。特别是近年来数据流分析方法在确认系统中也得到成功应用，用以查找如引用未定义变量等程序错误。也可以用来查找对以前未曾使用的变量再次赋值等数据流异常的情况。找出这些错误是很重要的，因为这常常是常见程序错误的表现形式，如错拼名字、名字混淆或是丢失了语句。这里将首先说明数据流分析的原理，然后指明它可揭示的程序错误。

如果程序中，某一语句执行时能改变某程序变量 V 的值，则称 V 是被该语句定义的。如果一语句的执行引用了内存中变量 V 的值，则说该语句引用变量 V。例如，语句

```
X := Y + Z
```

定义了 X，引用了 Y 和 Z，而语句

```
If Y > Z then goto exit
```

只是引用了 Y 和 Z。输入语句

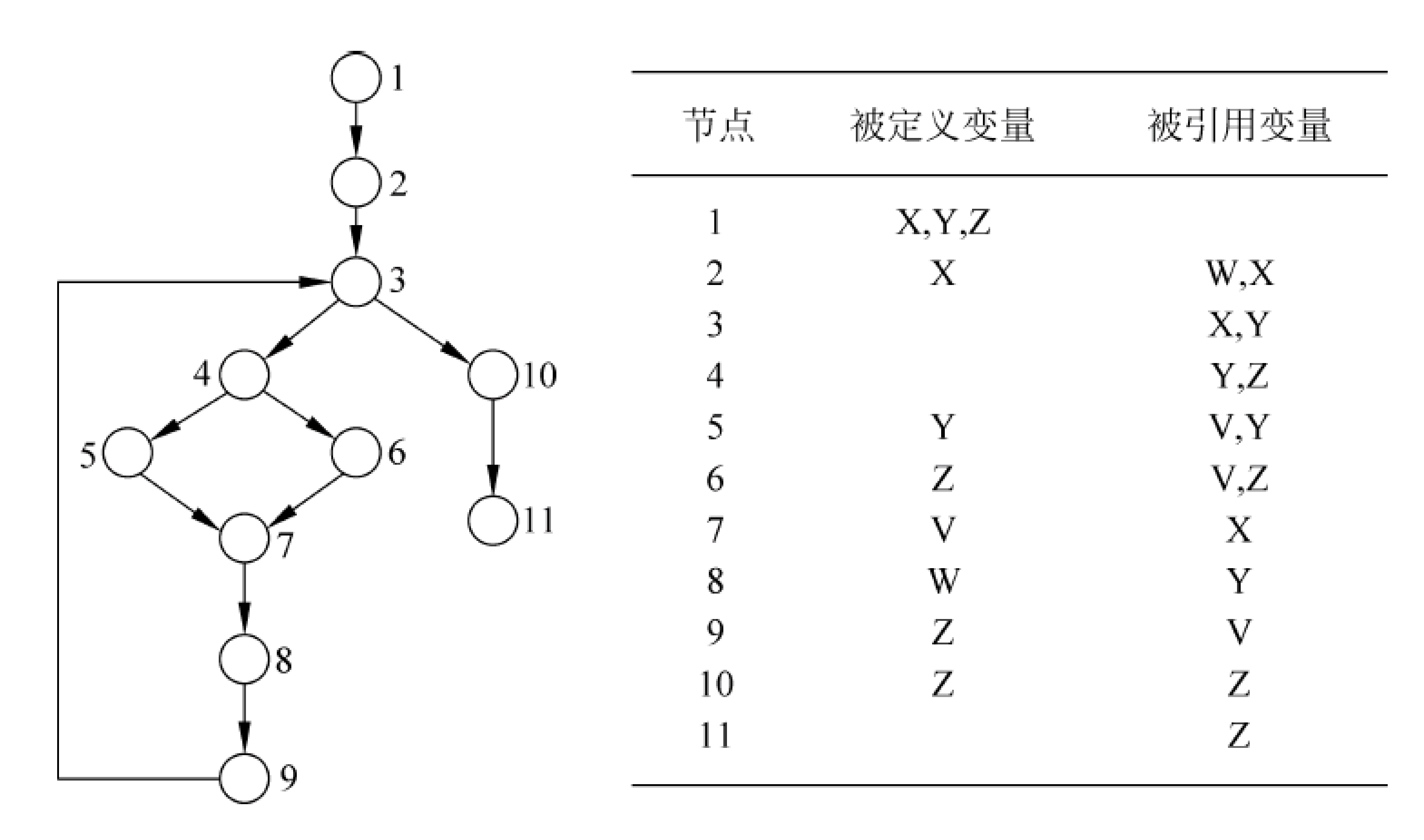
```
READ X
```

定义了 X。输出语句

```
WRITE X
```

引用了 X。执行某个语句也可能使变量失去定义，成为无意义的。例如，在 FORTRAN 中，循环语句 DO 的控制变量在经循环的正常出口离开循环时，就变成无意义的。

图 5-5 给出了一个小程序的控制流图,同时指明了每一语句定义和引用的变量。可以看出,第一个语句定义了 3 个变量 X、Y 和 Z,这表明它们的值是程序外赋给的。例如,该程序是以此三变量为输入参数的过程或子程序。同样,出口语句引用 Z 表明 Z 的值被送给外部环境。



节点	被定义变量	被引用变量
1	X,Y,Z	
2	X	W,X
3		X,Y
4		Y,Z
5	Y	V,Y
6	Z	V,Z
7	V	X
8	W	Y
9	Z	V
10	Z	Z
11		Z

图 5-5 控制流图及其定义和引用的变量

该程序中含有两个错误:

① 语句 2 使用了变量 W,而在此之前并未对其定义;

② 语句 5、6 使用了变量 V,这在第一次执行循环时也未对其定义过。

此外,该程序还包含两个异常:

① 语句 6 对 Z 的定义从未使用过;

② 语句 8 对 W 的定义也从未使用过。

当然,程序中包含一些异常,如 3、4 也会引起执行的错误。不过这一情况表明,也许程序中含有错误;也许可以把程序写得更容易理解,从而能够简化验证工作,以及随后的维护工作(去掉那些多余的语句一般会缩短执行时间,不过在此我们并不关心这些)。

总之,在数据流最初分析中,我们可以集中关注定义/引用异常的缺陷,如变量被定义,但是从来没有使用;所使用的变量没有被定义;变量在使用之前被定义了两次。另外,因为程序内的语句因变量的定义和使用而彼此相关,所以用数据流测试方法更能有效地发现软件缺陷。但是,在度量测试覆盖率和选择测试路径的时候,数据流测试很困难。

对数据流进行分析的工作可借助编译器或程序分析工具来完成。

5.2.5 代码安全性检查

所谓代码安全性,就是代码在运行时,或被调用时产生错误的容易程度。如 C++,它规定了严格的语法,然而又有其灵活性,然而这种灵活性增加了程序的不可预见性,这直接导致故障的产生,致使代码的安全性变差。譬如指针的指向发生错误,则直接导致结果错误;另外指针指向越界,可能导致缓冲区溢出,很多病毒就是利用缓冲区溢出对电脑进行攻击的;还有 C++提供的一些关于字符处理的函数,虽然提供了这方面的功能,但没有对参数范围进行限制和检查,这也很容易导致错误的发生。

代码安全性检查或静态错误分析主要用于确定在源程序中是否有某类错误或"危险/不安全"的结构。一般可借助工具来对代码的安全性进行检查。

1. 代码安全性检查方法

对代码进行安全性检查大致有四种方法。

1）对变量类型和单位进行检查

为了强化对源程序中数据类型的检查，发现在数据类型上的错误和单位上的不一致性，在程序设计语言中扩充了一些结构。如单位分析要求使用一种预处理器，它能够通过使用一般的组合/消去规则，确定表达式的单位。

2）对变量引用进行检查

最常用的代码安全性检查，方法就是发现引用异常。如果沿着程序的控制路径，变量在赋值以前被引用，或变量在赋值以后未被引用，这时就发生了引用异常。为了检测引用异常，需要检查通过程序的每一条路径。也可以建立引用异常的探测工具。

3）对表达式运算进行检查

对表达式进行检查，以发现和纠正在表达式中出现的错误。包括在表达式中不正确地使用了括号造成错误，数组下标越界造成错误，除数为 0 造成错误，对负数开平方或对 π 求正切值造成错误，以及对浮点数计算的误差。

4）接口分析

对接口进行安全性检查主要检查模块、过程、函数它们之间接口的一致性。因此要检查形参与实参在类型、数量、维数、顺序、使用上的一致性；检查全局变量和公共数据区在使用上的一致性。

2. 代码安全性检查关注的错误

具体而言，代码安全性检查要发现如下错误。

1）坏的存储分配

内存的申请和释放都有相应的函数，两者必须一一对应才能将申请的内存释放掉。用 maloc/caloc/realloc 申请的内存，对应的用 free 来释放；用 new 申请的内存，必须用 delete 来释放，如果对应不匹配，则发生错误。

2）内存泄漏

程序在执行过程中，在内存中动态申请的内存在函数返回或程序退出时必须有相应的释放操作，没有执行释放操作，将导致内存丢失，尤其在循环体中，变量的循环累积有可能造成系统崩溃。

3）指针引用

指针变量在使用前必须确保指向确定的地址单元，如果指针为空或指向错误的地方则会导致程序出现异常。

4）约束检查

越界指针、数组越界就是指针或数组超出了原先设定的范围，导致出现了意想不到的结果或异常。

5）变量未初始化

变量定义后必须被初始化，未初始化的变量的值不确定，使用它会使程序出现不正确的结果，甚至导致程序出现异常。

6）错误逻辑结构

检查在逻辑上可能有错误的结构以及多余的不可达的程序段(不可达代码是指永远执行不到的代码)。如交叉转入转出的循环语句，为循环控制变量赋值，存取其他模块的局部数据等。

7）其他

缓冲区溢出、非法类型转换、非法的算数运算（例如除零错误、负数开方）、整数和浮点数的上溢出/下溢出、多线程对未保护数据的访问冲突等都可能导致出现异常。

5.3　软件复杂性分析

软件危机产生的最直接原因是软件复杂性已远远超出人们对复杂性控制的能力。软件复杂性越高，软件中隐含错误的概率越大，软件的可靠性、可维护性越差。近几十年来的研究表明，作为软件显著特点的复杂性是导致软件错误的主要原因，软件可靠性问题的本质就是复杂性问题。软件的可靠性与其复杂性密不可分，当复杂性超过一定限度时，软件缺陷或错误便急剧上升，甚至引发软件开发的失败。此外，软件的可维护性等质量特性也与之有极大关系。1979 年，Belady 和 Lehman 提出的软件维护模型表明，软件的维护开销除与维护人员的素质等相关外。维护工作量是复杂性的一个指数函数。由此可见，软件复杂性度量与控制是软件可靠性工程亟待解决的一个重要问题。而对软件复杂性进行分析、度量和控制，其目的就是减少由软件设计方法和技巧使用不当而带来的复杂性，可以更好地对软件开发过程进行控制，并降低由复杂性引发软件错误的可能性，提高软件的可靠性和可维护性，确保软件产品的质量。

对于软件复杂性的研究已经有几十年的历史，人们对软件复杂性的基本认识是：越复杂的事物越容易出错，并带来问题。软件复杂性反映在分析、设计、测试、维护和修改软件的困难程度或复杂程度，而且这种复杂性将来会与日俱增。

软件复杂性产生的原因主要有如下几个方面：

（1）软件应用需求太复杂、应用要求太高。

（2）软件开发环境（包括编程、调试、测试、应用仿真等）及应用环境太复杂。

（3）软件应用框架、结构及模型太复杂。

（4）软件开发过程和开发模型太复杂。

（5）软件项目因涉及人的智力劳动其管理太复杂。

（6）软件设计与验证太复杂，尤其是网络分布式应用软件和嵌入式应用软件的设计与验证。

总之，软件复杂性是在软件开发过程中逐渐产生的，最终体现主要在软件结构复杂性和算法复杂性等方面。

5.3.1　软件复杂性度量与控制

软件复杂性度量是对软件复杂性的定量描述，是软件复杂性分析和控制的基础。软件复杂性度量的结果是软件复杂度。对象不同，描述软件复杂性的角度和方法不同。程序长度等一些经典方法曾发挥过积极的作用，但它们仅仅反映了软件规模，没有真实地反映软件的复杂性。于是，人们又根据软件结构，从数据流和控制流角度出发，结合软件的模块复杂性、结构复杂性及这两者的总体复杂性度量，来实现对软件复杂性的度量。这是一种从本质上反映软件复杂性的综合方法。不过，基于总体和全局的、能从本质上综合反映软件复杂性的精确度量方法尚待进一步研究。

目前，人们已经研究出来许多度量的方法和标准，但主要分为两大类：面向对象的软件复杂性度量和面向过程的软件复杂性度量。而研究最为活跃、成果也较多的是面向过程的软件复杂性度量。其中著名的软件复杂性度量方法有 Line Count 语句行度量、基于 FPA 功能点

分析的度量、Halstead 软件科学度量法和 McCabe 结构复杂性度量。随着近几年面向对象技术的兴起，面向对象分析（OOA）、面向对象设计（OOD）、面向对象编程（OOP）的技术方法和工具发展很快，且得到广泛应用。面向对象复杂性度量方法 C&K、MOOD 等的提出和应用标志着软件复杂性度量进入到面向对象度量的阶段。

虽然上述软件复杂性度量方法都有各自的缺点和不足，但在一定程度上，从不同的侧面反映了软件的复杂性，其中许多方法已成功地运用在软件开发中。

事实上，即便是最精确的度量，软件复杂性度量也只是为软件复杂性的定量分析和控制提供依据，其根本目的是通过控制软件复杂性来改善和提高软件的可靠性。设计过程是软件复杂性形成的根源，因此最有效的办法是在软件设计过程中对软件复杂性进行有效控制，使之保持在一个合理的范围内。

1. 软件复杂性度量

目前软件复杂性度量主要根据软件结构来实现对软件复杂性的度量，软件结构复杂性包含模块复杂性和所有模块结构复杂性两个方面。这是一种从本质上反映软件复杂性的方法。

1）模块复杂性

模块复杂性包含了两部分内容：模块内部结构复杂性和模块接口复杂性。因此，模块复杂性度量主要用来对模块中的数据流结构、控制流结构（或模块信息流结构）和模块之间互连的复杂程度等进行度量和评价。

模块复杂性度量力图反映模块内部结构复杂性，以及模块之间的调用关系（即接口复杂性）。其中，模块内部结构复杂性度量是软件复杂性度量的基础，是一种曾一度被广泛使用的传统方法，主要有 McCabe 度量和程序长度（Halstead）度量两种方法；模块接口复杂性将一个模块对应一个节点，节点之间的连接关系就是模块之间的调用关系，模块接口复杂性定义为以起点为顶点的有向图的所有路径数加 1。另外，模块接口复杂性也常用模块的扇入扇出数量或信息的扇入扇出数量来度量（模块的扇入等于进入该模块的信息流与该模块的输入数据之和，模块的扇出等于进入该模块的信息流与该模块的输出数据之和）。总之，模块复杂性度量是基于程序数据流、程序控制流和模块调用关系的有向图的拓扑结构的，但人们通常将模块接口复杂性称为模块结构复杂性。

模块结构复杂性度量的另一方面是试图反映包括所有模块接口关系在内的整个软件的结构复杂性。模块结构复杂性度量不仅反映了模块之间的接口情况，我们还可以通过它将已发现的失效和软件可靠性及与之相关的数据复杂性有机地联系起来。

对模块结构复杂性的度量，不论是静态的，还是动态的，都可以通过对模块或信息的扇入扇出数量来度量，即通过计算直接进入或流出模块的数据流路径来获得模块的扇入扇出数据。通常，还可以通过自动数据流层次或 HIPO 图等来确定模块之间、子系统之间或模块与子系统之间信息流的扇入扇出数。即接口或结构复杂性可表示成对应模块或子系统的扇入与扇出乘积的平方。

程序只有主模块、无子模块时的结构复杂性为 1。结构复杂性高的软件或模块比结构复杂性低的软件或模块的可靠性差。它将软件复杂性和软件可靠性有机地联系起来。当然，还可以考虑对程序长度加权。

2）总体复杂性度量

总体复杂性或整个软件结构的复杂性同模块复杂性之间往往相互矛盾。模块划分越小，

功能越简单,模块内部结构复杂性就越小;但模块之间的联系就越多,接口就越复杂,由此导致的模块结构复杂性就越大。相反,模块内部结构复杂性的增加意味着模块结构复杂性的降低。因此,软件总体复杂性是在软件设计,尤其是软件总体设计中,试图通过对总体复杂性的度量来综合反映软件的功能要求及软件中的模块划分情况等,并力求平衡模块复杂性与模块结构复杂性;寻找一个合适的复杂性指标,从而达到改善和提高软件总体复杂性的目的,使之最小化。总体复杂性是模块复杂性、模块结构复杂性,以及软件的重要度、调用频率等的综合。但遗憾的是,目前尚没有一种权威的总体复杂性度量方法。

2. 软件复杂性控制

软件复杂性控制通常是从模块内部结构复杂性控制和模块结构复杂性控制两方面进行。

1)模块内部结构复杂性控制

单个模块容易理解,可以分别编译、调试、查错、修改、测试和维护。它在容易理解和处理的同时,还可以有效地防止错误蔓延,从而降低软件复杂性,提高软件可靠性。因此,模块化是结构化程序设计的基础,是软件的主要属性,而模块复杂性控制是软件复杂性控制的基础。模块大小由其功能和性能决定,模块化不够或过分模块化都应得到有效的控制。模块内部结构复杂性控制主要包括模块的大小和结构的控制。

在三种典型的控制结构中,循环结构的复杂性最大。在模块内部,对循环结构复杂性的度量非常有意义。但是,循环复杂性的度量没有将连续程序语句、单独的程序语句或由多支路条件语句所引起的控制流复杂性计算在内,这无疑降低了度量的准确性。它的过度使用相当危险,通过增加模块数量来降低循环复杂性也不理智。

似乎是模块划分越小,由此产生的复杂性就越低,对应的软件可靠性就越高。如果无限地分割软件,最后形成的软件复杂性极小,可靠性水平无限高。而事实上,还有其他因素制约着软件的复杂性。随着模块数目的增加,尽管模块内部结构复杂性减小了,但模块之间的接口随之增加,甚至大幅度增加,从而导致结构复杂性增加。因此,应适当地确定模块的大小和接口,使之保持适度的复杂性,这是模块内部结构复杂性控制的基本原则。

一个软件最合适的模块数和最合适的模块大小目前尚无一种精确的指标。因此,在完成软件规定功能的前提下,在开发成本的有效控制下,寻找一种“最佳”的软件模块大小和数目,来达到对软件模块内部结构复杂性的控制。

为改善模块内部结构复杂性,提高软件可靠性,在进行模块设计时需遵循如下原则。

(1) 抽象化。对于复杂软件的模块化分解,常常采用抽象的层次分解。首先用一些高级的抽象概念进行构造和理解,这些高级的概念又可以用一些较低级的概念来构造和理解,如此进行下去,直至最低层次的具体元素。

(2) 模块化。模块化概念与抽象是一致的。随着软件设计的逐步推进,软件结构中每个模块层次代表了软件抽象层次的一次精化。事实上,软件结构的顶层模块控制了系统的主要功能,并且影响全局;软件结构的低层模块完成对数据的具体处理。用自顶向下、由抽象到具体的方式分配控制,从而精化软件设计,降低软件的复杂性,提高软件的可理解性、可测试性和可维护性。

(3) 信息隐蔽。在模块功能和大小的确定及设计过程中,遵循信息隐蔽原则不仅有利于改善模块本身的复杂性,也有利于降低软件的结构复杂性。信息隐蔽意味着有效的模块化可以通过定义一组独立的模块来实现,这些独立的模块彼此间仅交换那些为了完成软件功能而必须交换的信息。此外,在软件测试和维护过程中,信息隐蔽也将提供极大的好处。因为大多数数据和过程对软件的其他部分而言是隐蔽的。因此,即使在调试和维护期间由于疏忽而引

入了错误，一般也不会波及软件的其他部分。

2）模块结构复杂性控制

软件结构是软件元素之间的关系表示。软件元素之间的关系多种多样，这些关系均可表示为层次形式，即层次之间是由关系连接的，故受到关系的制约。这种层次结构的概念已经成为软件结构的一种表示形式。

同样为改善模块结构复杂性，提高整个软件可靠性，在进行软件体系结构设计时须遵循如下原则。

（1）努力使模块独立。开发具有单一功能且和其他模块之间没有更多相互作用的模块，使之保持独立性，是软件结构复杂性控制的基本要求。对模块化程度较高的软件，其功能可以被分割，接口进行简化。因此，这样的软件比较容易编制，尤其适用于不同成员分别编制。而且，独立的模块容易测试和维护，因为由设计和代码修改引起的副作用已得到限制，错误不容易蔓延。这样，有可能得到"插件式"模块。总之，模块独立性是优秀软件设计的关键。一般可通过对模块的耦合性和内聚性的控制，来实现对软件结构的控制。

耦合性表示模块之间的相对独立性，是影响软件复杂性的一个重要因素。设计软件时，如果我们遵循尽量使用数据耦合、少用控制耦合、限制公共耦合范围、完全不用内容耦合这 4 个原则，就可能在很大程度上减少模块之间的耦合性，降低模块的复杂性。

内聚是指模块功能的相对强度，用于衡量一个模块内部各个元素彼此结合的紧密程度，是信息隐蔽和局部化的自然延伸。在软件设计中，虽然没有必要精确地确定软件的内聚性等级，但必须力争使所设计的模块具有高内聚性，并能识别出低内聚性。即适当调整软件设计，以便得到更好的模块独立性。

（2）适当的扇入扇出。扇出是影响模块宽度的主要因素，扇出越大，模块越复杂。此时，应适当增加中间层次的控制模块，以降低模块的扇出。而扇出太小时，可把下级模块进一步分解成若干子功能模块或合并到它的上级模块中去。因此，一个模块的扇出不能太大，也不能太小。经验表明，典型系统的平均扇出通常是 3 或 4，上限是 5～9。模块的扇入越大，共享该模块的上级模块数就越多，这是有好处的。但是，不能违背模块独立性原则，而一味地追求高扇入。通常，软件结构对扇入扇出的要求是，顶层扇出较多，中间层次的扇出应尽量减少，随着深度增加，争取更多的扇入。

（3）简化软件接口。接口复杂性是产生错误的根源。在设计软件接口时，应尽量使软件接口传递的信息简单，并与模块的功能一致。通常，我们追求的设计是单入口、单出口模块。这样不仅可以有效地防止模块之间的内容耦合，同时也便于降低接口的复杂性和冗余度。此外，还有利于一致性的改善。设计软件接口应尽量避免模块之间的病态链接，杜绝转移进入或引用到一个模块的内部。

3）软件总体复杂性控制

软件总体复杂性控制是一个系统工程。它是在对软件的各种复杂性度量的基础上，在综合考虑软件开发成本、可靠性要求等一系列因素之后，对软件复杂性的一种平衡。尽管其首要任务是软件总体复杂性控制，但单纯从复杂性来看，可能有时达到的并不是最好的复杂性要求，却可能是一种最优的控制与选择。因此，软件总体复杂性控制不仅是一种技术，更是一种艺术。它需要的不仅是技术、方法和工具，它更需要软件设计人员的超水平智力发挥。

下述内容是形成软件复杂性的重要原因，它们构成了软件复杂性度量的基本度量准则集，是软件复杂性控制的基本出发点。

（1）控制结构和数据结构复杂的程序较复杂。

(2) 跳转语句使用不当的程序较复杂。

(3) 全局变量较多的程序较复杂。

(4) 按地址调用参数比按值调用参数较复杂。

(5) 模块及过程之间联系密切的程序较复杂。

(6) 嵌套深度越大,程序越复杂。

(7) 循环结构复杂性大于选择结构和顺序结构的复杂性。

(8) 宽度是软件复杂性的主要形成原因。

软件复杂性度量力图对这些准则进行定义和定量描述,找出影响和制约软件复杂性的所有关系。而软件复杂性控制试图在软件设计中,通过对所有影响软件复杂性、进而影响软件可靠性的因素进行控制,将它们限制在最小的范围内,以改善和提高软件可靠性。另外,在编程语言选择上,我们可以选择能从软件工程中获取最大好处的现代语言,如 Ada 语言,它很好地支持抽象化、模块化、信息隐藏以及能够降低耦合、增加内聚的模块独立等。

5.3.2 软件复杂性度量元

度量软件复杂性的度量元很多,主要分为如下几类:

(1) 规模,即总共的指令数,或源程序行数。

(2) 难度,通常由程序中出现的操作数的数目所决定的量来表示。

(3) 结构,通常用与程序结构有关的度量来表示。

(4) 智能度,即算法的难易程度。

软件复杂性直接关系到软件开发费用的多少、开发周期长短和软件内部潜伏错误的多少。同时它也是软件可理解性的另一种度量。

对软件复杂性进行度量要求满足以下假设:

(1) 它可以用来计算任何一个程序的复杂性。

(2) 对于不合理的程序,例如对于长度动态增长的程序,或者对于原则上无法排错的程序,不应当使用它进行复杂性计算。

(3) 如果程序中指令条数、附加存储量、计算时间增多,不会减少程序的复杂性。

1. 规模度量元(Line Count 复杂度)

基于规模度量程序的复杂性,最简单的方法就是统计程序的源代码行数。此方法基于两个前提:

① 程序复杂性随着程序规模的增加不均衡地增长;

② 控制程序规模的方法最好是分而治之,将一个大程序分解成若干个简单的可理解的程序段。

方法的基本考虑是统计一个程序模块的源代码行数目,并以源代码行数作为程序复杂性的度量。若设每行代码的出错率为每 100 行源程序中可能有的错误数目,例如每行代码的出错率为 1%,则是指每 100 行源程序中可能有一个错误。

一般程序出错率的估算范围是 0.04%～7%,即每 100 行源程序中可能存在 0.04～7 个错误。另外,每行代码的出错率与源程序行数之间不存在简单的线性关系。随着程序的增大,出错率以非线性方式增长。所以,代码行度量法只是一个简单、粗糙的估计方法,在实际应用中很少使用。

2. 难度度量元(Halstead 复杂度)

Halstead 的软件科学理论也许是“最著名的和最完全的(软件)复杂度的综合度量”,它是

软件科学提出的第一个计算机软件的分析“定律”。

Halstead 软件科学研究确定计算机软件开发中的一些定量规律，它采用以下一组基本的度量值，这些度量值通常在程序产生之后得出，或者在设计完成之后估算出。Halstead 根据程序中可执行代码行的操作符和操作数的数量来计算程序的复杂性。操作符和操作数量越大，程序结构就越复杂。如对于代码，可以统计它们的操作符和操作数，然后以此为基础，计算程序的长度和体积等。其中，操作符包括程序调用、数学运算符以及有关的分隔符等，操作数可以是常数和变量等。

在高级语言定义中，运算符包括算术运算符、赋值符（＝或：＝）、逻辑运算符、分界符（，或；或：）、关系运算符、括号运算符，以及子程序调用符、数组操作符、循环操作符等；特别还有成对的运算符，例如，begin…end、for…to、repeat…until、while…do、if…then…else、（…）等都当做单一运算符。

设 n1 表示程序中不同运算符的个数，n2 表示程序中不同操作数的个数，N1 表示程序中实际运算符的总数，N2 表示程序中实际操作数的总数，参见表 5-1。

表 5-1 FORTRAN 语言程序的操作符、操作数计算举例

SUBROUTINE SORT(X,N)	操作符	计数	操作符	计数
DIMENSION X(N)				
IF(N. LT. 2)RETURN	1 语句末	7	1X	6
D0 20 I=2,N	2 数组下标	6	2 I	5
D0 10 J=1,I	3 =	5	3 J	4
IF (X(I) . GE. X(J)) GO TO 10	4 IF( )	2	4 N	2
SAVE=X(J)	5 DO	2	5 2	2
X(I)=X(J)	6 ,	2	6 SAVE	2
X(J)=SAVE	7 程序末	1	7 1	1
10 CONTINUE	8 . LE.	1	n2 = 7	N2 = 22
20 CONTINUE	9 . GE.	1		
RETURN	10 GO TO	1		
END	n1 = 10	N1 = 28		

Halstead 的程序词汇表为一个程序中出现的不同操作符和不同操作数之和：$n=n1+n2$；

实际的 Halstead 长度，即 N 表示实际的程序长度或简单长度，其定义为：$N=N1+N2$。

我们以 N ^ 表示程序的预测长度，Halstead 给出 N ^ 的计算公式为：$N\hat{}=n1\log_2 n1+n2\log_2 n2$。

Halstead 的重要结论之一是：程序的实际长度 N 与预测长度 N ^非常接近，这表明即使程序还未编写完也能预先估算出程序的实际长度 N。

Halstead 还给出了另外一些计算公式：

（1）程序容量：$V=N\log_2(n1+n2)$。

它表明了程序在词汇上的复杂性，其最小值为 $V\hat{}=(2+n2\hat{})*\log_2(2+n2\hat{})V$，这里，2 表明程序中至少有两个运算符：赋值符＝和函数调用符 f()，n2 ^表示输入/输出变量个数。

对于上面的例子，利用 n1，N1，n2，N2，可以计算得

$$N\hat{}=10*\log_2(2\times10)+7*\log_2(2\times7)=52.87,\quad N=28+22=50,$$

$$V=(28+22)*\log_2(10+7)=204$$

等效的汇编语言程序的 V＝328。这说明汇编语言比 FORTRAN 语言需要更多的信息量

（以 bit 表示）。

（2）程序级别：L^=(2/n1) * (n2/N2)。

它表明了一个程序的最紧凑形式的程序量与实际程序量之比，反映了程序的效率。其倒数 D=1/L^，表明了实现算法的困难程度。

（3）编制程序所用的工作量：E=V/L^。

（4）智能级别：I=L^ * E。

（5）语言级别：L'=L^ * L^ * V。

（6）编程时间（以小时记）：T^=E/(S * f)，这里 S=60 * 60，f=8。

（7）平均语句大小：N/语句数。

（8）程序中的错误数目预测值：$B=N\log_2(n1+n2)/3000$。B 为该程序的错误数。它表明程序中可能存在的差错 B 应与程序量 V 成正比。

例如，一个程序对 75 个数据库项共访问 1300 次，对 150 个运算符共使用了 1200 次，那么预测该程序的错误数：

$$B=(1200+1300)*\log_2(75+150)/3000\approx 6.5$$

即预测该程序中可能包含 6～7 个错误。

Halstead 的优点是：

① 不用对程序进行深层次的分析，就能够预测错误率，预测维护工作量；

② 有利于项目规划，衡量所有程序的复杂度；

③ 计算方法简单；

④ 与所用的高级程序设计语言类型无关。

众多的研究表明 Halstead 复杂度用于预测程序工作计划和程序错误有一定的意义。但 Halstead 复杂度仅考虑程序数据量和程序体积而不考虑程序控制流，不能从根本上反映程序复杂性。

3. 结构度量元（McCabe 复杂度）

软件复杂性主要表现为软件结构的复杂性，McCabe 复杂度是对软件结构进行严格的数学计算得来的，实质上是对程序拓扑结构复杂性的度量，明确指出了程序的复杂部分。McCabe 复杂度包括圈复杂度、基本复杂度、模块设计复杂度、设计复杂度、集成复杂度、行数、规范化复杂度、全局数据复杂度、局部数据复杂度、病态数据复杂度。

在软件工程中，有三种使用 McCabe 复杂性度量的方式。

（1）作为测试的辅助工具。McCabe 复杂性度量的结果等于通过一个子程序的路径数，因而需要设计同样多的测试用例以覆盖所有路径。如果测试用例数小于复杂性数，则有三种情况：

① 需要更多的测试。

② 某些判断点可以去掉。

③ 某些判断点可用插入式代码替换。

（2）作为程序设计和管理指南。在软件开发中，需要一种简单的方式指出可能出问题的子程序。保持子程序简单的通用方法是设置一个长度限制，例如 50 行或 2 页，但这实际上是在缺乏测试简明性的有效方法时无可奈何的替代方法。不少人认为 McCabe 度量就是这样一种简明性度量。但是要注意，McCabe 度量数大的程序，不见得结构化就不好。例如，Case 语句是良好的简单结构，但可能有很大的 McCabe 度量数，这可能是由于问题和解决方案所固有的复杂性所决定的。使用者应当自己决定如何使用 McCabe 度量所提供的信息。

（3）作为网络复杂性度量的一种方法。Hall 和 Preiser 提出了一种组合网络复杂性度量，用于度量可能由多个程序员在组网过程中按模块化原理建立的大型软件系统的复杂性。

1）McCabe 圈复杂度

McCabe 程序结构的复杂性主要指模块内部程序的复杂性。McCabe 度量方法根据图论和程序结构控制理论，当度量程序结构的复杂性时，首先把程序结构的控制流程图转化为有向图（即程序图），然后计算强连通有向图的环数来衡量软件的质量。用 McCabe 度量方法得到的复杂度称为程序结构的圈复杂度。为了方便叙述，我们给出以下几个定义。

定义 1：程序的流程图是被简化以后的，把程序流程图中每个处理框都退化成一个节点，流程图中的箭头转化为连接不同点的有向弧，由此得到的有向图称为程序图。

定义 2：强连通图是指从图中任一个节点出发都能到达所有的其他节点。

定义 3：强连通有向图的环数是指在一个强连通有向图中线性无关环的个数，即有向图 G 中的弧数 m 与节点数 n 的差再加上分离部分的数目 p，计算公式为 $V(G)=m-n+p$。

圈复杂度除了上述计算方法外，还有其他方法，如：圈复杂度等于程序图中判定节点的数目加 1；圈复杂度等于强连通程序图在平面上围成的区域个数。

图 5-6 是一个把考试成绩达到 80 分的学生的学号和成绩打印出来的程序控制流图，把它转换为程序图后，如图 5-7 所示。

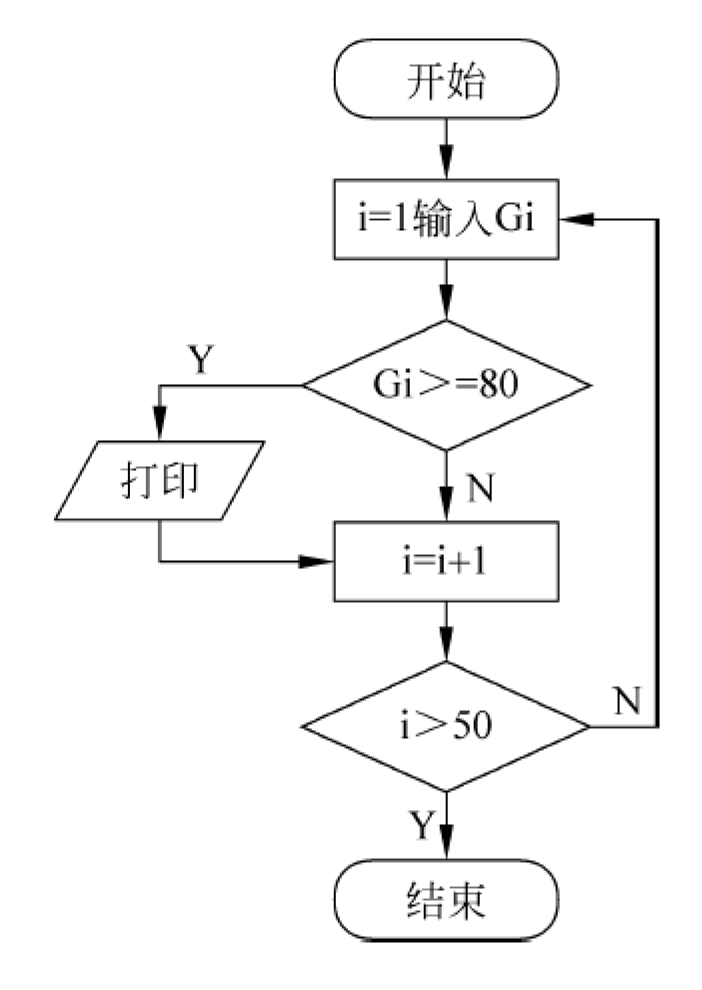


图 5-6 程序控制流程图

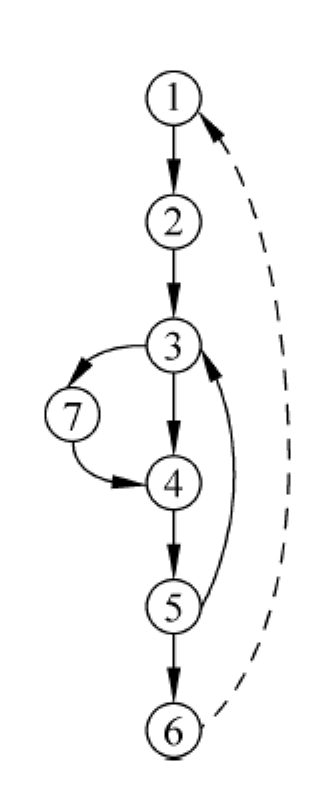


图 5-7 程序图

从图 5-7 中可知其不是强连通图，需从节点 6 到节点 1 添加一条虚弧线。由此，程序图中有 9 条弧，节点数为 7，根据公式 $V(G)=m-n+p$ 可得出环形复杂度为 3；程序图中判定节点是节点 3 和节点 5，用判定节点数目加 1，同样可以得到圈复杂度为 3；加上新添的虚弧线，根据程序图中围成的区域个数，同样也可以得到圈复杂度为 3。

圈复杂度数量上表现为独立路径的条数，即合理地预防错误所需测试的最少路径条数，圈复杂度大说明程序代码可能质量低且难于测试和维护，经验表明，程序的可能错误和高的圈复杂度有着很大关系。

McCabe 圈复杂度的优点是：

① 避免软件中的错误倾向；

② 指出极复杂模块，这样的模块也许可以进一步细化；

③ 度量测试计划，确定测试重点；

④ 指导开发人员限制程序逻辑，指导测试人员开展测试活动；

⑤ 指出将要测试的区域；

⑥ 帮助测试人员确定测试和维护对象；

⑦ 与所用的高级程序设计语言类型无关。

McCabe 圈复杂度应用：圈复杂度指出为了确保软件质量应该检测的最少基本路径的数目。在实际中，测试每一条路径是不现实的，测试难度随着路径的增加而增加。但测试基本路径对衡量代码复杂度的合理性是很必要的。

实践表明，模块的圈复杂度，即模块复杂性和模块中存在的软件错误数或缺陷数，以及为了发现并改正它们所需的时间之和，存在一种明显的关系。McCabe 指出，$V(G)$可用做最大模块复杂性的定量指标。通过大量软件工程数据研究发现，圈复杂度到 10 是模块复杂性的实际上限，当 $V(G)$超过这个值时，因为高的圈复杂度使测试变得更加复杂而且增大了软件错误产生的概率。

圈复杂度是以软件的结构或控制流程图为基础进行度量的。控制流程图描述了软件模块的逻辑结构。一个模块在典型的语言中是一个函数或子程序，有一个入口和一个或多个出口，也可以通过调用/返回机制设计模块。软件模块的每个执行路径，都有与从模块的控制流程图中的入口到出口的节点相符合的路径。

对圈复杂度的限制：现在有许多好方法可以用来限制圈复杂度。过于复杂的模块容易出错，难以理解、测试、更正，所以应当在软件开发的各个阶段有意识地限制复杂度，许多开发者已经成功地实现把对软件复杂度的限制作为软件项目的一部分，尽管在确切的数目上略微有些争议。最初支持的数目是 10，现在支持的数目可达 15。但是，只应当在条件较好的情况下使数目大于 10，例如开发者非常有经验，设计合乎正式标准，使用现代化的程序语言、结构程序、代码预排和先进的测试计划。换句话说，开发团队可以选择超过 10 的限制数目，但是必须根据经验进行一些取舍，把精力放在比较复杂的模块上。

McCabe 度量方法反映了代码的质量，预测代码维护量，辅助模块划分。McCabe 度量方法与所有的高级程序设计语言类型无关。但是其实质上是对程序控制流复杂性的度量，并不考虑数据流，因而其科学性和严密性具有一定的局限性。

2）Essential Complexity (ev(G))基本复杂度

基本复杂度是用来衡量程序非结构化程度的。非结构成分降低了程序的质量，增加了代码的维护难度，使程序难于理解。因此，基本复杂度高意味着非结构化程度高，难以模块化和维护。

基本复杂度的计算方法是将圈复杂度图中的结构化部分简化成一个点，计算简化以后流程图的圈复杂度就是基本复杂度。其优点是：

（1）衡量非结构化程度。

（2）反映代码的质量。

（3）预测代码维护量，辅助模块划分。

（4）与所用的高级程序设计语言类型无关。

基本复杂度的应用方法是：

（1）当基本复杂度为 1，这个模块是充分结构化的。

（2）当基本复杂度大于 1 而小于圈复杂度，这个模块是部分结构化的。

（3）当基本复杂度等于圈复杂度，这个模块是完全非结构化的。

3）Module Design Complexity (iv(G))模块设计复杂度

模块设计复杂度是用来衡量模块判定结构，即模块和其他模块的调用关系。软件模块设

计复杂度高意味模块耦合度高，这将导致模块难于隔离、维护和重用。

模块设计复杂度的计算方法是从模块流程图中移去那些不包含调用子模块的判定和循环结构后得出的圈复杂度，因此模块设计复杂度不能大于圈复杂度，通常是远小于圈复杂度。

模块设计复杂度用于：

① 衡量模块对其下层模块的支配作用。

② 衡量一个模块到其子模块进行集成测试的最小数量。

③ 定位可能多余的代码。

④ 以复杂的计算逻辑和设计来区分模块。

⑤ 是设计复杂度(S0)和集成复杂度(S1)计算的基础。

⑥ 与所用的高级程序设计语言类型无关。

4）Design Complexity (S0)设计复杂度

设计复杂度用来衡量程序模块之间的相互作用关系，给出了系统级模块设计复杂度的定性描述，有助于衡量自底向上集成测试的效果。另外，它提供了全面衡量程序设计规格和复杂度的数据，不用关注独立模块的内部情况。高设计复杂度的系统意味着系统各部分之间有着复杂的相互关系，这样系统将难以维护。

S0 是程序中所有模块设计复杂度之和，可应用于完整的软件，也可应用于任何子系统。包括：

① 衡量代码的质量。

② 指出一个模块整体的复杂度，反映了每个模块和其内部模块的控制关系。

③ 揭示了程序中模块调用的复杂度。

④ 有助于集成复杂度的计算。

5）Integration Complexity (S1)集成复杂度

集成复杂度是为了防止错误所必须进行的集成测试的数量表示，或者说是程序中独立线性子树的数目，一棵子树是一个有返回的调用序列。就像圈复杂度是测试路径的数目，而集成复杂度是程序或其子系统的独立线性子树。

集成复杂度的计算方法是，一个程序的集成复杂度和一个模块的圈复杂度是非常相似的，必须计算对程序进行完全测试所需集成测试的数目。S1 的计算公式：$S1=S0-N+1$，N 是程序中模块的数目。

集成复杂度计算有助于集成测试的实施，量化集成测试工作且反映了系统设计复杂度，有助于从整体上隔离系统复杂度。

6）McCabe 的其他复杂度计算

McCabe 的 Number of Lines(nl)行数是模块中总的行数，包括代码和注释。

McCabe 的 Normalized Complexity (nv)规范化复杂度是圈复杂度除以行数，它与所用的高级程序设计语言类型无关。规范化复杂度定义那些有着显著判定逻辑密度的模块，这些模块相对于其他常见规范模块需要做更多的维护工作。

McCabe 的 Global Data Complexity (gdv(G))全局数据复杂度量化了模块结构和全局数据变量的关系，它说明了模块对外部数据的依赖程度，同时度量了全局数据的测试工作，也描述了模块之间的耦合关系，能反映潜在的维护问题。

McCabe 的 Specified Data Complexity (sdv(G))局部数据复杂度量化了模块结构和用户局部数据变量的关系，同时度量了局部数据的测试工作。

McCabe 的 Pathological Complexity (pv(G))病态复杂度衡量一个模块包含的完全非结

构化成分的程度,标出向循环内部跳入的问题代码,而这些部分具有最大的风险度,通常需要重新设计。其计算方法是:所有的非结构部分除去向循环内跳入的结构,转化为线性结构,病态复杂度就等于简化以后流程图的圈复杂度。它与所用的高级程序设计语言类型无关,指出了可靠性的问题,降低了维护风险,帮助人们识别极不可靠的软件。

4. 其他度量元

除了前面介绍的 Line Count、Halstead 和 McCabe 三种复杂性度量外,对模块复杂性、模块结构复杂性进行度量还有很多度量元,如函数参数个数、路径数、层次数、直接调用个数、RETURN 语句个数、调用者的个数、GOTO 语句个数、词汇频度、局部变量个数、注释率、函数中的可执行语句数、宏定义个数、扇入扇出数等。

5.3.3 面向对象的软件复杂性度量

随着面向对象技术和工具的发展日益成熟,面向对象设计显示了其巨大的优越性。各种运用面向对象技术分析、设计的软件系统越来越多,传统的软件度量方法无法适应面向对象技术中采用的数据抽象、封装、继承、多态性、信息隐藏、重用等机制,这些机制在传统的面向过程的软件开发中是缺乏的,因此,运用传统的度量方法不能很好地反映面向对象技术的特征,如:继承提供的重用,属于对象自身的代码较少,如果用源代码行(LOC)来度量整个对象显然结果是不能令人满意的;如果仅把类看做模块也是不恰当的,因为功能性的模块之间的耦合关系表现在接口上,主要是参数传递、对全程变量的访问以及模块间的调用关系。而对象间的耦合关系主要表现为通过继承、通过消息传递和接收、通过对抽象数据类型的引用带来的耦合。传统的度量方法无法反映这些特征,因此,需要额外的度量方法(即面向对象度量方法)来反映这些关系。

与传统软件一样,面向对象度量的主要目的是:更好地理解产品的质量,评价过程的效率,改进项目完成工作的质量。

1. 面向对象度量的特性

任何产品的技术度量都取决于产品的特性。面向对象软件与使用传统方法开发的软件的度量方法截然不同,目前面向对象软件度量主要针对以下 5 个特性。

1) 局域性

局域性(Localization)是指信息被集中在一个程序内的方式。

(1) 传统方法:数据与过程分离,功能分解和功能信息局域化。其典型的实现形式为过程模块,工作时用数据驱动功能。

此时的度量放在功能内部结构的复杂性上(如模块规模、聚合度、圈复杂度等)或放在该功能与其他功能(模块)的耦合方式上(接口复杂性)。

(2) 面向对象方法:局域性基于对象,因为类是面向对象系统的基本单元,对象封装数据和过程,因此应把类(对象)作为一个完整实体来度量。

另外,操作(功能)和类之间的关联是一对一的。因此,在考虑类协作的度量时,必须能适应一对多和多对一的关联。

2) 封装性

封装(Encapsulation)是指一个项集合的包装。

(1) 传统方法:记录、数组,只有数据没有过程,为低层次的封装;过程、函数、子例程和段,则只有过程没有数据,为中层次的封装。其度量的重点分别在代码行的数据和圈复杂度。

(2) 面向对象方法:面向对象系统封装拥有类的职责(操作),包括类的属性、操作和特定

的类属性值定义的类(对象)的状态。其度量和重点不是单一的模块,而是包含数据(属性)和过程(操作)的包。

3) 信息隐藏

信息隐藏(Information Hiding)是指隐藏了程序体实现的细节,只将访问该程序所必要的信息提供给访问该程序的其他程序。

在这一点上,面向对象方法和传统方法基本一致。因此,面向对象系统应支持信息隐藏,除提供隐藏等级说明的度量外,还应提供面向对象设计质量指标。

4) 继承性

继承性(Inheritance)是指一个对象的属性和操作能够传递给其他对象的机制。继承性的发生贯穿于一个类的所有层次。

一般来说,传统软件不支持这种特性。而对面向对象系统来说,继承性是一个关键特性。因此,很多面向对象系统的度量都以此为重点,如子的数量(类的直接实例数量)、父的数量(直接上一代数量),以及类的嵌套层次(在一个继承层次中,类的深度)。

5) 抽象

抽象(Abstraction)使设计者只关心一个程序的主要细节(数据和过程两者),而不考虑底层的细节。

抽象也是一种相对概念,抽象在面向对象和传统开发方法中都被采用,如处于抽象的较高层次时,可忽略更多的细节,只提供一个关于概念或项的一般看法;当处于抽象的较低层次时,可以引入更多的细节,即提供一个关于概念或项的更详细的看法及具体实现。

在面向对象中,由于类是一个抽象,它可以从许多不同的细节层次和用许多不同的方式(如作为一个操作的列表、一个状态的序列、一组协作)来观察。

面向对象度量可用一个类度量的项来表示抽象,如每个应用类的实例化的数量、每个应用类被参数化的数量,以及类被参数化与未被参数化的比例等。

2. 面向类的度量

类是面向对象系统的基本单元。对类进行度量,可用于评价面向对象软件的设计质量。

第一位以类为对象开展面向对象度量方法研究的是 Chidamber 和 Kemerer(C&K 或 CK)。CK 度量是使用最为广泛的度量体系之一,他们建议使用 6 种基于类设计的度量(通称为 CK 度量组):

① 每个类的加权方法数。

② 继承树的深度。

③ 孩子的个数。

④ 对象类之间的耦合。

⑤ 类的响应。

⑥ 方法中的聚合缺乏。

与此同时,Lorenz 和 Kidd 提出了 LK 度量组,他们把基于类的度量分为四种类型:规模、继承、内部(特性)和外部(特性)。对面向对象类进行基于规模的度量,主要集中在单一类的属性和操作的数量,以及作为整个面向对象系统的平均值;基于继承的度量,关注的是贯穿于类层次的操作被重用的方式;类的内部特性的度量是考察聚合和代码问题;而外部特性的度量则是检查耦合和重用问题。

由于 CK 度量方法和 LK 度量方法的出发点是类这一级,虽对系统开发有帮助,却没有提供对一个系统的判断。为此,Abrito 等人针对面向对象属性提出了一套称为 MOOD 的度

量。MOOD度量则是系统级的,它从封装性、继承性、耦合性和多态性四方面提出了度量指标。

1) CK度量组

CK度量组的6种基于类设计的度量元详细描述如下。

(1) 每个类的加权方法(Weighed Methods per Class,WMC)。

$$\text{WMC} = \Sigma C_i (i = 1 \sim n)$$

其中,C_i为一个类的各个方法(或操作或服务)的复杂性,相当于传统方法中的圈复杂度,C_i可相加。方法的数量和它们的复杂性给出了用来实现和测试一个类的工作总量。方法的数量越大,继承树(所有子类都继承父类的方法)就越复杂。对一个给定的类,随着方法的数量增大,其应用很可能变得越来越专门化,由此将限制其可能的重用。所以,WMC的值应当合理。

(2) 树的深度(Depth of the Inheritance Tree,DIT)。

这种度量被定义为从节点到树根的最大长度。DIT的值越大,复杂性就越高。因为随着DIT的增大,层次的类可能会继承许多方法。当试图预测一个类行为时,困难不仅会增大,而且会增加设计的复杂性。当然DIT较大时,则表示有许多方法被重用,这是其好的一方面。

(3) 孩子的个数(Number Of Children,NOC)。

子类在类的层次内,子类可以最直接地从属于一类。随着子类数量的增大,重用也增加了。但父类抽象的表示可能减少,即一些子类可能不是父类真正的成员,同时,测试数量(用来检查每个子类在操作前后的要求)也将增加。

(4) 对象类之间的耦合(Coupling Between Object Classes,CBO)。

CBO是指一个类合作(即相关)的数量。当CBO增大时,不仅降低了可重用性,而且使其修改和修改后的测试变得复杂。所以,每个类的CBO值应当保持合理。这与在传统软件中减少耦合的一般原则是一致的。

(5) 类的响应(Response For a Class,RFC)。

一个类的响应设置是一组方法,它可能被执行,用来响应接收到的类对象的消息,RFC被定义为响应设置方法的数量。RFC增加,测试序列增加,测试工作量也将增加。由此可以得出,当RFC增大时,类的设计复杂性也将增大。

(6) 方法中的聚合缺乏(Lack of COhesion in Methods,LCOM)。

一个类内的每种方法访问一个或多个属性(也称实例变量)。LCOM是访问一个或多个相同属性方法的数量。如果LCOM很大,则说明方法可以通过属性与其他方法耦合,这就增加了类设计的复杂性。通常,对LCOM值很大的类,可以把它分为两个或多个单独的类,这样每个类的设计更方便。

这里讲的耦合和聚合与传统软件中所讲的是一样的。我们希望高聚合和低耦合,即保持低的LCOM。但在某些情况下,LCOM值很大也是合理的。

2) LK度量组

LK度量组是把基于类的度量分为四种类型:规模、继承、内部(特性)和外部(特性)。LK有如下4个度量元。

(1) 类大小(Class Size,CS)。

类的整体大小可用被封装在类中的操作的总数(包括继承来的和私有的操作)和被封装在类中的属性的数量(包括继承来的和私有的属性)度量来确定。CK度量组中提出的WMC度量也是类大小的加权度量。因此,类比的结果是:大的CS值指明类可能有太多的任务,它将

减少重用性并使实现和测试复杂化。通常，在确定类的大小时，继承的及公共的操作和属性应该加大加权力度，私有的操作和属性造成特例化并在设计中要求更加局部化。

类属性和操作数量的平均值也可以被计算。平均值越低，系统中类的重用性越好。

(2) 由子类重载的操作数量(Number of Operations Overridden by a subclass，NOO)。

存在这样一种情形，子类用自己使用的特殊版本替换了从其超类继承的某操作，这称为重置。大的 NOO 值通常指明了某种设计问题，如：因为子类应该是其父类的特殊化，它应该主要扩展父类的操作，这将导致独特的新的方法名。

如果 NOO 是大的，则设计者违反了父类所蕴涵的抽象，这导致了弱的类层次和可能难于测试和修改的 OO 软件。

(3) 由子类加入的操作的数量(Number of Operations Added by a subclass，NOA)。

子类通过加入私有的操作和属性而特例化。当 NOA 的值增大时，则子类产生了对父类隐含抽象的变异。通常，当类层次的深度增加(DIT 变大时)，在层次中低层的 NOA 值将下降。

(4) 特例化指标(Specialization Index，SI)。

特例化指标提供了对 OO 系统中每个子类的特例化程度的粗略指示。

特例化可通过加入或删除操作或通过覆写来达到。

$$SI = [NOO \times level]/M_{total}$$

其中 level 是类驻留在类层次中的级别或层次，M_{total} 是类的方法的总数，SI 的值越高，越有可能在类层次中包含了更多的与从父类抽象的类相比发生了变化的类。

3) MOOD 度量组

MOOD 度量是另一个著名的度量体系，它基于类的方法、属性等从封装性、继承性、耦合性和多态性四个方面给出了 OO 软件的六个度量指标，它是从 OO 系统的角度对程序的 OO 特征进行度量。

(1) 封装性度量。

由于封装性是通过类中的属性和方法实现的，因此对封装性的度量可以通过对属性隐蔽因子 AHF(Attribute Hiding Factor)和方法隐蔽因子 MHF(Method Hiding Factor)进行度量来表示。其中 MHF 的定义如下：

$$MHF = \sum_{i=1}^{TC}\sum_{m=1}^{Md(C_i)}(1 - V(M_{mi}))/\sum_{i=1}^{TC} Md(C_i)$$

这里 $Md(C_i)$是类 C_i，$i=1,\cdots,n$ 中定义的方法数目，TC 是类的总数目。

$$V(M_{mi}) = \sum_{j=1}^{TC} is_visiable(M_i, C_j)/TC - 1$$

在 $j \neq i$ 且 C_j 可调用 Mmi 时，$is_visible(M_{mi}, C_j)=1$。否则 $is_visible(M_{mi}, C_j)=0$。

AHF 也是以同样的方法定义的，只是使用属性，而不是方法。

数据封装(简单地说封装)经常被用于衡量一个语言隐藏具体实现的能力。通常我们可以用信息隐藏来定义。度量信息隐藏的属性是代码的可见性(code visibility)。AHF 和 MHF 使用类的属性和方法对其他类的代码的可见性来度量信息隐藏。

(2) 继承性度量。

MOOD 对继承性的度量也可以通过属性继承因子 AIF(Attribute Inheritance Factor)和方法继承因子 MIF(Method Inheritance Factor)进行度量来表示。

方法继承因子(MIF)是 OO 系统的类体系结构针对方法(操作)和属性而使用继承的程

度，它被定义为 $MIF = \Sigma Mi(C_i)/\Sigma Ma(C_i)$，　对 i 从 1 到 TC 求和。
其中 TC 为在体系结构中的类的总数，C_i 是在体系结构中的一个类。且：

$$Ma(C_i) = Md(C_i) + Mi(C_i)$$

其中 $Ma(C_i)$为在和 C_i 关联中可被调用的方法的数量，$Md(C_i)$为在类 C_i 中声明的方法的数量，$Mi(C_i)$为在类 C_i中继承(未被复写的)的方法的数量。

MIF 值提供了继承对 OO 软件的影响的指示。它表示系统中所有类的方法继承的程度，其值在 0～1 之间。若 MIF 越大，则系统中方法继承的程度越高。

AIF 也是用同样的方法来定义的，它表示系统中所有类的属性继承的程度，其值为 0～1。若 AIF 越大，则系统中属性继承的程度越高。

(3) 耦合性度量。

M00D 方法中使用耦合因子 CF(Coupling Factor)来度量类之间的耦合。CF 定义如下：

$$CF = \sum_{i=1}^{TC}\sum_{j=1}^{TC} is_client(C_i, C_j)/(TC^2 - TC)$$

这里针对 i 从 1 到 TC 和 j 从 1 到 TC 求和。当且仅当在客户端类 C_c 和服务器类 C_s间存在关系，且 $C_c \neq C_s$ 时，函数 is_client=1。否则，is_client=0。

CF 的计算考虑到了所有可能的类对集合，判断每对类之间是否存在通过消息传递或通过语义联系(一个类的方法或属性被另一个类引用)产生的耦合。就耦合来说，这两种耦合关系是相等的。

CF 表示系统中类间的耦合程度，其值为 0～1。一个系统的 CF 的值，对于衡量系统的复杂性、封装性，可重用性、可理解性和可维护性具有重要意义。可以凭直觉判断出 CF 值越大，类间发生的耦合越频繁，从而造成系统的封装性越差，重用的可能性越低，可理解程度越低，维护的困难越大。但我们很难说，耦合度越高，系统变得越复杂。例如，我们很容易构造一个耦合度很高的简单系统。大量的经验数据在这方面可能会有助于我们分析 CF 值的范围对于这些外在属性的影响。

(4) 多态性度量。

MOOD 度量方法中用多态因子 PF (Polymorphism Factor)来度量系统中存在多态的可能性。PF 定义为：

$$PF = \sum_{i=1}^{TC} Mo(C_i) / \sum_{i=1}^{TC} [Mn(C_i) * DC(C_i)],$$　这里对 i 从 1 到 TC 求和。且

$$Md(C_i) = Mn(C_i) + Mo(C_i)$$

其中，$Mn(C_i)$为新方法的数量，$Mo(C_i)$为覆写方法的数量，$DC(C_i)$为后代计数(某基类的后代类的数量)。

PF 的计算是重新定义继承的方法数目除以可能出现多态情况(子孙类的新方法是重载的)的最大方法数目得出的。因而，PF 反映了一个系统的动态连接(dynamic binding)情况，即系统中所有类的方法使用多态机制的程度，其值为 0～1。若 PF 越大，则类之间发生耦合就会越频繁。

4) CK、LK 及 MOOD 度量组的比较

上述三种基于类的度量方法中的度量元都反映出面向对象技术的特点，但度量的侧重点有所不同。如：

(1) CK 度量组中，系统的多态性通过 RFC 和 WMC 被间接地度量，CBO 和 LCOM 评测封装性，系统的继承性用 DIT 和 NOC 评测，但它没有多态性度量指标，是令人遗憾的方面；

(2) LK 度量组中，CS 反应系统的封装性，NOO 是基于继承的度量(它着重于在整个类层

次中操作被重用的方式),NOA考察内聚和面向源代码的问题,SI检查耦合和复用;

(3) MOOD度量方法则从封装性、继承性、耦合性、多态性四个方面给出六个度量指标,作用于类的属性和方法。

另外考虑的度量角度是有区别的,如:耦合性度量(CBO和CF)非常相似,CBO提供了从类一级考虑的耦合,但CF则是系统级的观点。因此,可以认为CK度量指标是从类一级出发考虑的,它对系统设计和开发大有帮助,而MOOD度量则提供了一个对系统的判断,它对工程管理者较为有用。

3. 面向操作的度量

因为类是OO系统设计中的基础框架或模板设施,而OO系统真正活动的实体是类的对象。类对象具有状态和行为两大特性,分别用数据和操作表示。因此对类操作进行复杂性度量是很有意义的。根据Lorenz和Kidd的建议,对类操作进行复杂性度量有三种度量元。

1) 平均操作规模(average Operation Size,OSavg)

虽然代码行数可以成为操作规模的衡量指标,但代码行数的度量与传统软件一样有许多问题。因此,在OO软件中,由操作所传送的消息数量提供了一个对操作规模度量可选的方法。操作规模增大,表示操作所传送的消息数量增加,数量越大,说明越复杂。

2) 操作复杂性(Operation Complexity,OC)

一个操作的复杂性可以用传统软件所使用的任何复杂性度量进行计算。因为操作限于特定的职责,所以设计人员应使OC尽可能地小。

3) 每个操作参数的平均数(average Number per oPeration,NPavg)

操作参数的数量越大,对象间的合作就越复杂。所以,NPavg一般应尽可能地小。

这里所说的三种面向操作的度量列出了对OO的操作可提供的几个度量元,通过对操作的度量来评价操作、操作传递的消息及对象间协作的复杂性,可以对整个系统的操作的复杂程度有所了解,但并不能代表整个软件系统的复杂度,还需从多个角度综合考虑。

4. 对OO系统的度量

根据Binder的建议,按对封装性和继承性的影响提出了几类对OO系统的度量方法。

1) 封装性

(1) 方法(操作与服务)中聚合的缺乏(LCOM)。LCOM的变量值越高,表示更多的状态必须进行测试,才能保证方法不产生副作用。

(2) 公共与私有的百分比(Percent Public and Protected,PAP)。公共属性从其他类继承,所以这些类是可见的。私有属性是专属的,为一特定子类所拥有。这种度量说明类的公共属性的百分比,PAP的值高就可能增加类间的副作用。

(3) 数据成员的公共访问(Public Access to Data Member,PAD)。这种度量说明可访问其他类属性的类的数量,即一种封装的破坏。PAD的值高可能导致类间的副作用。所以,测试的设计必须保证每一种这样的副作用能够被发现。

2) 继承性

(1) 根类的数量(Number of Root Classes,NOR)。这种度量是在设计模型中,描述性质各不相同的类层次数量的计算方法。对每一个根类及其子类的层次必须开发相应的测试序列。随着NOR的增大,测试工作量也相应增加。

(2) 扇入(Fan In,FIN)。当扇入用于OO情况时,扇入是一种多重继承的指标。FIN大于1,说明一个类不只从属一个根类,而是继承更多的根类的属性和操作。

(3) 孩子的数量(Number of Children,NOC)和继承树的深度(Depth of the Inheritance

Tree,DIT)。父类的方法(操作、服务)发生变化将需对每个子类进行重新测试。

5.4　软件质量模型

软件质量是软件开发工作的关键问题,也是软件生产中的核心问题。因为软件质量不高是导致软件项目进度延误、预算超支或项目失败、项目终止等软件危机的根本原因。另外,软件质量高可以降低项目开发的总成本,如:降低维护成本(并延长软件的生命期),降低软件失效导致的成本损失。最后,我们可以通过减少并纠正实际的软件开发过程和软件开发结果与预期的软件开发过程和软件开发结果的不符合情况,通过在软件开发周期中尽可能早地预期或检测到不符合的情况,防止错误发生,减少错误纠正成本等手段来保证软件质量。但在开展这些工作之前,我们必须回答什么是软件质量。

5.4.1　软件质量概念

软件质量是一种模糊而又捉摸不定的概念。我们在日常生活中常听到的关于软件评价有:这个软件真好用;这个软件功能全面、结构合理、简单易用。这些很平常的语言根本不能算作软件质量的评价,尤其不能算作软件质量科学的定量评价。

不同的人从不同的角度来看软件质量问题,会有不同的理解。从用户的角度看,质量就是满足客户的需求;从开发者的角度看,质量就是与需求说明保持一致;从产品的角度看,质量就是产品的内在特点;从价值的角度看,质量就是客户是否愿意购买。

现代质量管理认为,质量是客户要求或者期望的有关产品或者服务的一组特性,落实到软件上,这些特性可以是软件的功能、性能和安全性等。这些特性决定了软件产品保证客户满意的能力,并且,这些特性应该是可以度量的。虽然软件的无形性和复杂性使得软件质量的度量要比其他产品,比如电视机,困难得多,但我们仍可以借助软件测试的理论、技术、方法和工具来获得软件质量客观的、科学的度量。

另外,我们还可以从另一个角度,即软件产品是如何生产出来的,来间接地推断软件质量。我们称之为软件的流程质量,以有别于前面所说的软件产品质量。所谓流程,我们可以将其理解为一个活动序列和与此相关的输入、输出、约束条件、实现方法、辅助工具等因素共同组成的系统。ISO9001 和 SW-CMM 都主要是从流程角度来探讨软件质量和质量改进的。

当然,我们还能从其他角度,比如软件的生产者——人的素质,来诠释软件质量,但不管怎样,软件的产品质量是最终的检验标准,而最终的检验者就是客户。从这个意义上说,软件质量就是客户满意度。但这种说法太笼统,不好掌握。为此,我们必须给软件质量一个明确的概念或定义,以帮助我们有一个明确的尺度来检验质量。

1. 软件质量定义

国际标准化组织(ISO)在质量特性国际标准 ISO/IEC 9126 中将软件质量定义为反映软件产品满足规定需求和潜在需求能力的特征和特性的总和。MJ. Fisher 将软件质量定义为:所有描述计算机优秀程度的特性的组合。也就是说为了满足软件的各项精确定义的功能、性能要求,符合文档化的开发标准,需要相应地给出或设计一些质量特性及其组合,要得到高质量的软件产品,就必须使这些质量特性得到满足。

按照 ANSI/IEEE Std 1061—1992 中的标准,软件质量定义为:与软件产品满足需求所规定的和隐含的能力有关的特征或特性的全体。具体包括:

(1) 软件产品中所能满足用户给定需求的全部特性的集合。

(2) 软件具有所有的各种属性组合的程度。

(3) 用户主观得出的软件是否满足其综合期望的程度。

(4) 决定所用软件在使用中将满足其综合期望程度的综合特性。

通过类比，我们这样理解软件质量：软件质量是许多质量属性的综合体现，各种质量属性反映了软件质量的方方面面。人们通过改善软件的各种质量属性，从而提高软件的整体质量(否则无从下手)。

2. 软件质量特性

对于软件质量有三种不同的视面。用户主要感兴趣的是如何使用软件、软件性能的情况和使用软件的效果。所以他们关心的是：

① 是否具有所需要的功能。

② 可靠程度如何。

③ 效率如何。

④ 使用是否方便。

⑤ 环境开放的程度如何(即对环境、平台的限制，与其他软件连接的限制)。

而开发者负责生产出满足质量要求的软件，所以他们关心的是中间产品的质量以及最终产品。对于管理者来说，更注重总的质量，而不是某一特性。

从管理角度对软件质量进行度量所关心的那些影响软件质量的主要因素，可划分为三组，分别反映用户在使用软件产品时的三种观点。即：

① 正确性、健壮性、效率、完整性、可用性、风险(产品运行)。

② 可理解性、可维修性、灵活性、可测试性(产品修改)。

③ 可移植性、可重用性、互运行性(产品转移)。

而按照 ISO/IEC 9126 的规定，软件质量可用 6 个特性来评价。

(1) 功能性(functionality)：是与一组功能及其指定的性质有关的一组属性。这里的功能是指满足明确或隐含要求的那些功能。而这组属性以软件为满足需求做些什么来描述，而其他属性则以何时做和如何做来描述。

(2) 可靠性(reliability)：是与在规定的一段时间和条件下，软件维持其性能水平的能力有关的一组属性。在这里要强调的是：软件不会老化。因此可靠性的种种局限是由于需求、设计和实现中的错误所致，由这些错误引起的故障取决于软件产品使用方式和程序任选项的选用方法，而不取决于时间的流逝。

(3) 可用性(usability)：是与一组规定或潜在用户为使用软件所需作的努力和对这样的使用所作的评价有关的一组属性。这里软件的可用性主要是指人们学习、操作、准备输入和解释程序输出(输出结果和出错信息)的便利程度。反映了与用户的友善性，即用户在使用本软件时是否方便。

(4) 效率(efficiency)：在规定条件下，相对于所用资源数量，软件产品提供适当性能的能力。效率反映了在完成功能要求时，有没有浪费资源(资源这个术语有比较广泛的含义，它包括了内存、外存的使用，通道能力及处理时间等)。

(5) 可维护性(maintainability)：可维护性反映了在用户需求改变或软件环境发生变更时，对软件系统进行相应修改的容易程度。软件产品可被修改的容易程度或能力，包括为了适应环境的变化和需求、功能规格说明的变化而对软件进行的修改、改进或更改。

(6) 可移植性(portability)：软件产品从一个计算机系统/环境转移到另一个计算机系统/环境的容易程度。这里的环境可包括系统体系结构环境、硬件或软件环境。

显然，软件质量所反映的问题包括如下几个方面：

(1) 软件需求是度量软件质量的基础。

(2) 软件人员必须用工程化的方法来开发软件，否则软件质量就得不到保证(要根据所指定的标准来定义一组指导软件开发的准则。如果不能遵守这些准则，就极有可能导致质量不高)。

(3) 软件要满足一些隐含的需求，如可维护性、可靠性等。

总之，计算机软件质量是计算机软件内在属性的组合，包括计算机程序、数据、文件等多方面的可理解性、正确性、可用性、可移植性、可维护性、可修改性、可测试性、灵活性、重用性、完整性、适用性、健壮性、可靠性、效率与风险等多方面特性。是与软件产品满足规定的和隐含的需求的能力有关的特性和特性的全体，包括明确声明的功能和性能需求，明确文档化过程的开发标准，而且由专业人员开发的软件应具有的所有隐含特性都得到满足。对于一个高质量的软件，能够按照预期的时间和成本提交给用户，并能够按照预期要求正确工作的软件。

5.4.2　软件质量分层模型

尽管软件质量是难于定量度量的软件属性，但人们还是想尽办法来定性和定量地描述软件质量。依上所述，软件质量可用特定的软件质量特性来表示，而软件质量特性反映了软件的本质。讨论一个软件的质量，问题最终要归结到定义软件的质量特性。而定义一个软件的质量，就等价于为该软件定义一系列质量特性。软件质量特性是面向管理的观点，或者是从使用者的观点引入，体现了用户想要什么样的软件。

人们通常用软件质量模型来描述影响软件质量的特性。目前，已有多种有关软件质量的模型，它们共同的特点是基于软件质量特性定义软件质量分层模型。一般是定义为三层模型，参见图5-8。在这三层模型中，软件质量不仅从该软件外部表现出来的特性来确定，而且必须从其内部所具有的特性来确定；第二层的质量子特性是上层质量特性的细化，一个特定的子特性可以对应若干个质量特性。软件质量子特性是管理人员和技术人员关于软件质量问题通信渠道；下层是软件质量度量因子或度量元(包括各种参数)，用来度量质量特性。定量化的度量元可以直接测量或统计得到，为最终得到软件质量子特性值和特性值提供依据。

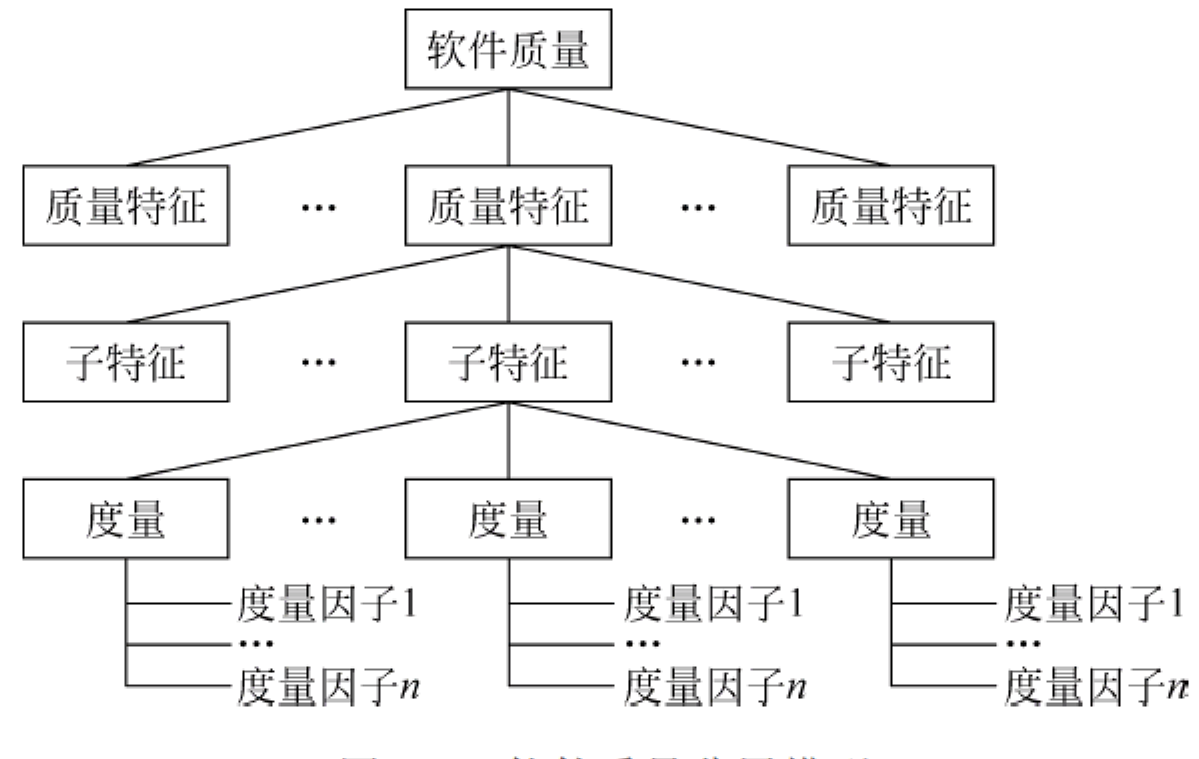


图5-8　软件质量分层模型

目前，已有很多质量模型，它们分别定义了不同的软件质量属性。比较常见的三个质量模型是McCall模型(1977年)、Boehm模型(1978年)和ISO/IEC 9126模型(1993年)。下面介绍几个影响较大的软件质量模型。

1. McCall 质量模型

McCall 等人将软件质量分解至能够度量的层次，提出 FCM 三层模型，即软件质量要素(factor)、衡量标准(criteria)和度量标准(metrics)，见表 5-2。

表 5-2 McCall 的 FCM 三层质量模型

层 级	名 称	内 容
第一层	质量要素：描述和评价软件质量的一组属性	功能性、可靠性、可用性、效率性、可维护性、可移植性等质量特性以及将质量特性细化产生的副特性
第二层	衡量标准：衡量标准的组合反映某一软件质量要素	精确性、稳健性、安全性、通信有效性、处理有效性、设备有效性、可操作性、培训性、完备性、一致性、可追踪性、可见性、硬件系统无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、自描述性、简单性、结构性、文件完备性等
第三层	度量标准：可由各使用单位自定义	根据软件的需求分析、概要设计、详细设计、编码、测试、确认、维护与使用等阶段，针对每一个阶段制定问卷表，以此实现软件开发过程的质量度量

在 FCM 三层模型中，软件质量概念是基于 11 个基本特性之上，而这 11 个特性分别面向软件产品的产品操作(product operation)、产品修正(product revision)和产品转移(product transition)。

(1) 正确性：一个程序满足它的需求规格说明和实现用户任务目标的程度。

(2) 可靠性：一个程序满足所需的精确度并最终完成它的预期功能的程度。

(3) 效率：一个程序完成其功能所需的计算资源和代码的度量。

(4) 完整：对未授权人员访问软件或数据的可控制程度。

(5) 可用性：学习、操作、准备输入和解释程序输出所需的工作量。

(6) 可维护性：定位和修复程序中一个错误所需的工作量。

(7) 灵活性：修改一个运行的程序所需的工作量。

(8) 可测试性：测试一个程序以确保它完成所期望的功能所需的工作量。

(9) 可移植性：把一个程序从一个硬件或一个软件系统环境移植到另一个环境所需的工作量。

(10) 可重用性：一个程序可以在另外一个应用程序中重用的程度。

(11) 互连性：连接一个系统和另一个系统所需的工作量。

McCall 将这 11 类软件质量特性，分为三类质量要素，它们之间的关系如图 5-9 所示。

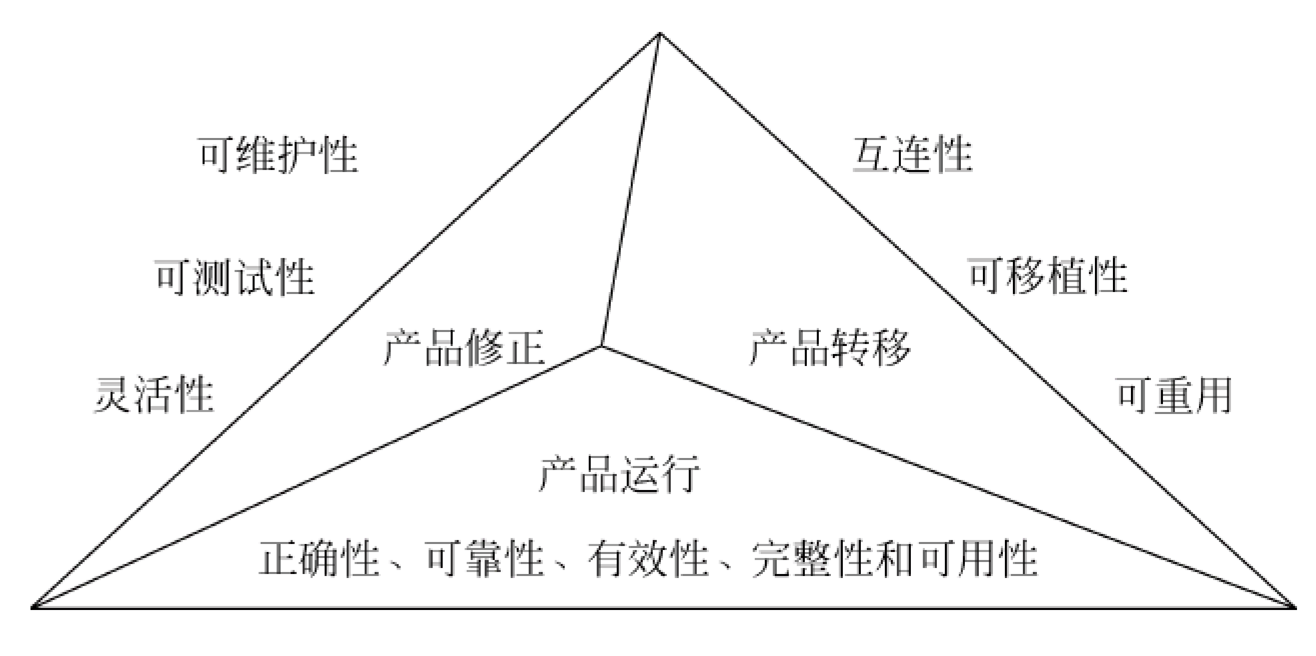


图 5-9 McCall 模型

第一类质量要素表现软件的运行特征，包括正确性、可靠性、有效性、完整性和可用性。第二类质量要素表现软件承受修改的能力，包括可维护性、灵活性、可测试性。第三类质量要素表现软件对新环境的适应程序，包括可移植性、可重用性、可互操作性。

McCall 等在 FCM 三层模型基础上又给出了一个三层次的质量度量框架，如图 5-10 所示。McCall 等人认为，要素是软件质量的反映，软件属性可用做评价的准则，定量化地度量软件属性可知软件质量的优劣。

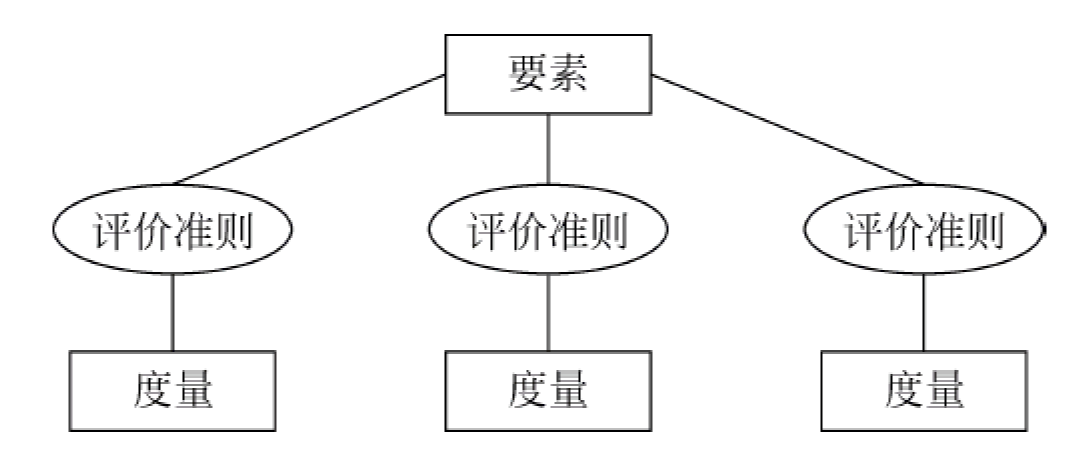


图 5-10 McCall 度量模型框架

McCall 定义的 21 种软件质量要素评价准则如下所示。

(1) 可审查性(audit ability)：检查软件需求、规格说明、标准、过程、指令、代码及合同是否一致的难易程序。

(2) 准确性(accuracy)：计算和控制的精度，是对无误差程序的一种定量估计，最好表示成相对误差的函数。值越大表示精度越高。

(3) 通信通用性(communication commonality)：使用标准接口、协议和频带的程序。

(4) 完全性(completeness)：软件系统不丢失任何重要成分，完全实现系统所需功能的程度。

(5) 简明性(conciseness)：程序源代码的紧凑性。

(6) 一致性(consistency)：在软件开发项目中一致的设计和文档技术的使用。

(7) 数据通用性(data commonality)：在程序中使用标准的数据结构和类型。

(8) 容错性(error-tolerance)：系统在各种异常条件下提供继续操作的能力。

(9) 执行效率(execution efficiency)：程序运行效率。

(10) 可扩充性(expandability)：能够对结构设计、数据设计和过程设计进行扩充的程度。

(11) 通用性(generality)：程序部件潜在的应用范围的广泛性。

(12) 硬件独立性(hardware independence)：软件同支持它运行的硬件系统不相关的程度。

(13) 检测性(instrumentation)：监视程序的运行，一旦发生错误时，标识错误的程序。

(14) 模块化(modularity)：程序部件的功能独立性。

(15) 可操作性(operability)：操作一个软件的难易程度。

(16) 安全性(security)：控制或保护程序和数据不受破坏的机制，以防止程序和数据受到意外的或蓄意的存取、使用、修改、毁坏或泄密。

(17) 自文档化(self-documentation)：源代码提供有意义文档的程度。

(18) 简单性(simplicity)：理解程序的难易程度。

(19) 软件系统独立性(software system independence)：程序与非标准的程序设计语言特征、操作系统特征以及其他环境约束无关的程度。

(20) 可追踪性(traceability)：从一个设计表示或实际程序构件中追溯需求的能力。

(21) 易培训性(training)：软件支持新用户使用该系统的能力。

2. Boehm 质量模型

Boehm 在 1976 年首次提出了软件质量层次模型，认为软件产品的质量基本上可从软件的可用性、软件的可维护性和软件的可移植性三个方面来考虑。并将软件质量在概念上分解为若干层次，对于最低层软件质量概念引入量化指标，以便得到软件质量的整体评价。

Boehm 在软件质量层次模型中的第一层，同样给出了功能性、可靠性、可用性、效率、可维护性和可移植性 6 个质量特性。

Boehm 在软件质量层次模型中的第二层给出了 22 个软件质量评价准则：精确性（在计算和输出时所需精度的软件属性）、健壮性（在发生意外时，能继续执行和恢复系统的软件属性）、安全性（防止软件受到意外或蓄意的存取、使用、修改、毁坏或泄密的软件属性），以及通信有效性、处理有效性、设备有效性、可操作性、培训性、完备性、一致性、可追踪性、可见性、硬件系统无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、自描述性、简单性、结构性、产品文件完备性。

Boehm 质量模型第一层与第二层的关系如图 5-11 所示。

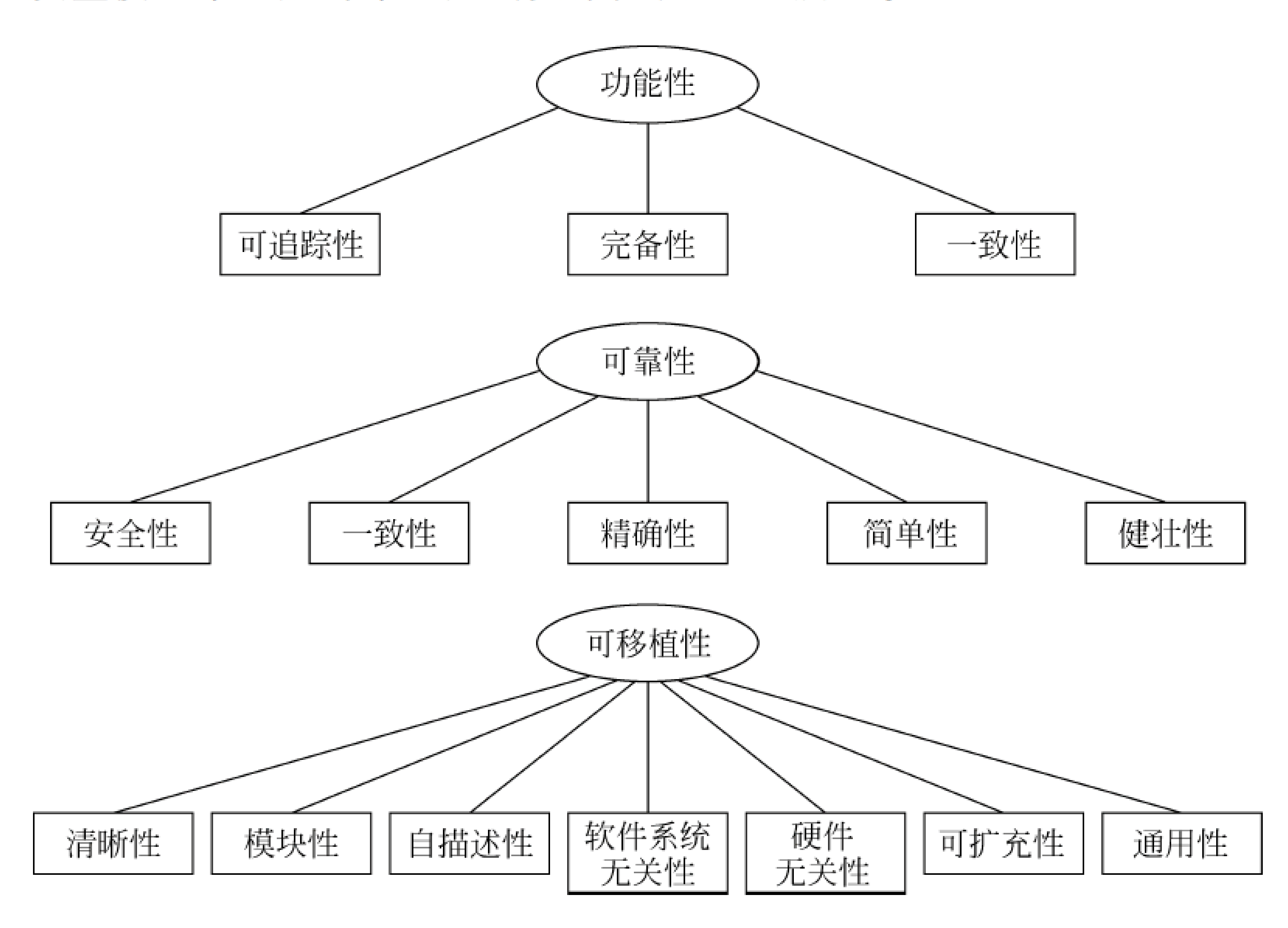


图 5-11　Boehm 质量模型第一层与第二层的关系

Boehm 在软件质量层次模型中的第三层是软件质量度量。根据软件的需求分析、概要设计、详细设计、实现、组装测试、确认测试和维护与使用七个阶段，制定了针对每一个阶段的问卷表，以此实现软件开发过程的质量控制。

对于企业来说，不管是定制，还是外购软件后的二次开发，了解和监控软件开发过程每一个环节的进展情况、产品水平都是至关重要的，因为软件质量的高低，很大程度上取决于用户的参与程度。

应用 Boehm 模型进行软件质量评价要注意以下几点。

（1）对于不同类型的软件，系统软件、控制软件、管理软件、CAD 软件、教育软件、网络软件以及不同规模的软件，其质量要求、评价准则、度量问题的侧重点有所不同应加以区别，如表 5-3 所示。

表 5-3 软件质量评价的着重点

应用环境特征	需要考虑的要素	应用环境特征	需要考虑的要素
生存期长	可移植性、可维护性	要在不同的环境中使用	可移植性
实时系统	可靠性、效率	在银行系统中使用	可靠性、功能性

(2) 在需求分析、概要设计、详细设计及其实现阶段，主要评价软件需求是否完备，设计是否完全反映了需求，以及编码是否简洁、清晰。而且，每一个阶段都存在一份特定的度量工作表，它由特定的度量元素组成，根据度量元素的得分就可逐步得到度量准则及质量要素的得分，并在此基础上做出评价。

(3) 对软件各阶段都进行质量度量的根本目的是以此控制软件成本、开发进度，改善软件开发的效率和质量。

3. ISO/IEC 9126 质量模型

ISO/IEC 9126 将软件质量定义为前面所介绍的六大特性：功能性、可靠性、可用性、效率、可维护性和可移植性，每个特性包括一系列子特性。

(1) 软件的功能性主要从三个方面考察。首先该软件产品的功能是否满足需求；其次现有功能是否达到设计要求；最后，所有功能是否正常实现。

(2) 软件的可靠性进一步定义了成熟性(Maturity)、容错性(Fault Tolerance)、易恢复性(Recover Ability) 3 个子特性。

(3) 软件的可用性进一步定义了可理解性(Understand Ability)、易学性(Learn Ability)、可操作性(Operability) 3 个子特性。

(4) 软件的效率进一步定义了时间特性(Time Behaviour)和资源特性(Resource Behaviour) 2 个子特性。

(5) 软件的可维护性进一步定义了易分析性(Analys Ability)、易改变性(Change Ability)、稳定性(Stability)、易测试性(Test Ability) 4 个子特性。

(6) 软件的可移植性进一步定义了适应性(Adapt Ability)、易安装性(Install Ability)、遵循性(Conformance)、易替换性(Replace Ability) 4 个子特性。

4. 质量模型 GB/T 16260—2006

我国 2006 年颁布的《信息技术 软件产品评价质量特性及其使用指南》GB/T 16260—2006 在 ISO/IEC 9126 质量模型基础上对软件质量从 6 个质量特性和 27 个质量子特性进行概念性的描述。图 5-12 给出了质量特性与质量子特性之间的关系。表 5-4 给出了质量特性与质量子特性的描述。

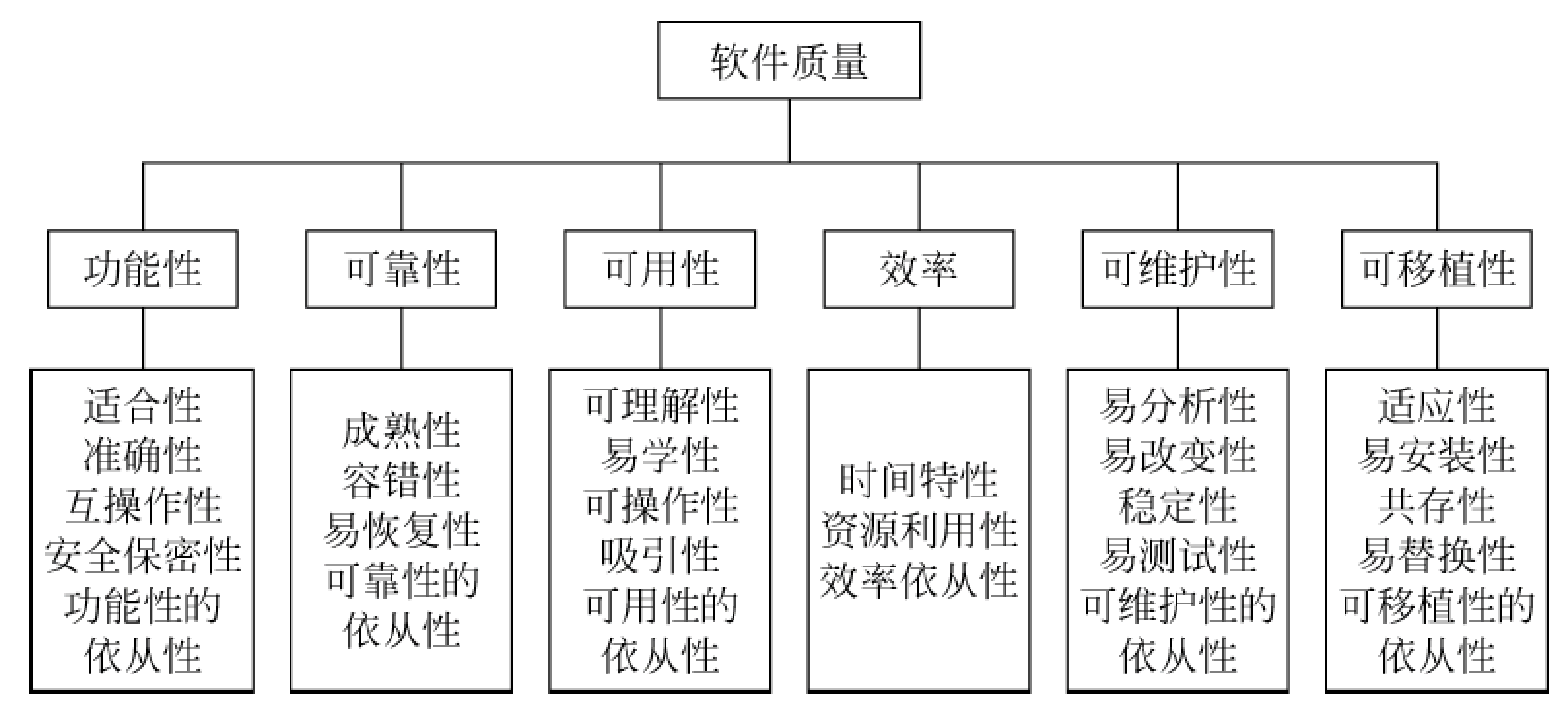


图 5-12 GB/T 16260—2006 的质量模型

表 5-4 质量特性与质量子特性的描述

质量特性	描　　述	质量子特性	质量子特性描述
功能性	与一组功能及其指定的性质有关的一组属性。这里的功能是指满足明确或隐含的需求的那些功能	适合性	与规定任务能否提供一组功能及这组功能的适合程度有关的软件属性
		准确性	与能否得到正确或相符的结果或效果有关的软件属性
		互操作性	与其他指定系统进行交互的能力有关的软件属性
		安全保密性	与防止对程序及数据的非授权的故意或意外访问的能力有关的软件属性
		依从性	是软件遵循有关的软件标准、约定和法规及类似规定的软件属性
可靠性	与在规定的一段时间和条件下,软件维持其性能水平的能力有关的一组属性	成熟性	与由软件故障引起失效的频度有关的软件属性
		容错性	与在软件故障或违反指定接口的情况下,维持规定的性能水平的能力有关的软件属性
		可恢复性	与在失效发生后,重建其性能水平并恢复直接受影响数据的能力,以及为达此目的所需的时间和能力有关的软件属性
可用性	与一组规定或潜在的用户为使用软件所需作的努力和对这样的使用所做的评价有关的一组属性	可理解性	与用户为认识逻辑概念及其应用范围所花的努力有关的软件属性
		易学性	与用户为学习软件应用所花的努力有关的软件属性
		可操作性	与用户为操作和运行控制所花努力有关的软件属性
效率	与在规定的条件下,软件的性能水平与所使用的资源量之间关系有关的一组属性	时间特性	与软件执行其功能时响应和处理时间及吞吐量有关的软件属性
		资源特性	与软件执行其功能时所使用的资源数量及其使用时间有关的软件属性
可维护性	与进行指定的修改所需的努力有关的一组属性	可分析性	与为诊断缺陷或失效原因及为判定待修改的部分所需努力有关的软件属性
		可修改性	与进行修改、排除错误或适应环境变化所需努力有关的软件属性
		稳定性	与修改所造成的未预料结果的风险有关的软件属性
		可测试性	与确认已修改软件所需的努力有关的软件属性
可移植性	与软件可从某一环境转移到另一环境的能力有关的一组属性	适应性	与软件无须采用有别于为该软件准备的活动或手段就可能适应不同的规定环境有关的软件属性
		易安装性	与在指定环境下安装软件所需努力有关的软件属性
		一致性	使软件遵循与可移植性有关的标准或约定的软件属性
		可替换性	与软件在该软件环境中用来替代指定的其他软件的机会和努力有关的软件属性

注意:表中的依从性实质上分别对应着功能性的依从性(软件产品遵循与功能性相关的标准、约定或法规以及类似规定的能力)、可靠性的依从性(软件产品遵循与可靠性相关的标准、约定或法规的能力)、可用性的依从性(软件产品遵循与可用性相关的标准、约定、风格指南或法规的能力)、效率依从性(软件产品遵循与效率相关的标准或约定的能力)、维护性的依从性(软件产品遵循与维护性相关的标准或约定的能力)、可移植性的依从性(软件产品遵循与可移植性相关的标准或约定的能力)。

从上我们看出，软件质量特性/子特性之间存在相互冲突，GB/T 16260—2006 的质量模型是面向所有软件的，因此它的质量属性面面俱到。但是对于一个具体的软件产品或软件项目来说，我们必须考虑利弊，全面权衡，根据质量需求，适当合理地选择/设计质量特性，并进行评价。标准中规定的质量特性、子特性、度量元不一定都要涉及，也就是说要根据软件产品本身的特点、领域、规模等因素来选择标准中的质量特性、子特性建立自己的质量模型，其中包括度量元的确定。关于度量元的确定可以从标准中选取，也可以根据实际情况补充若干度量元（因为标准中的度量元不是完备的），但体系最好与标准一致，即要有名称、度量目的、公式、指标、标度类型等内容。

5.4.3　软件质量度量与评价

软件产品的质量无法像硬件那样用很多技术指标来衡量，如重量、体积、温度系数等，但也不能用功能齐全、结构合理、层次分明、执行正确、符合用户规定的功能来概括，因为软件产品满足这些是远远不够的。另外，在软件项目的开发过程中，往往强调软件必须完成的功能、进度计划、花费成本，而忽略软件生命周期中各阶段的质量标准。最后，我们在评价软件质量时应强调软件总体质量（低成本、高质量），而不应片面强调软件正确性，忽略其可维护性与可靠性、可用性与效率等。因此，应在软件生命周期的各个阶段都注意软件的质量，而不能只在软件最终产品验收时注意质量；我们应制定软件质量标准，定量地评价软件质量，使软件产品评价走上评测结合，以测为主的科学轨道。

下面从软件生命周期上简述各个阶段应该考虑的评价准则，如表 5-5 所示。

表 5-5　软件生命周期各阶段质量的评价准则

开发阶段	评价准则
系统需求分析和设计	完备性、处理有效性、设备有效性、可操作性、培训性、一致性、可追踪性、可见性、硬件环境无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、简单性、结构性
软件需求分析	完备性、精确性、处理有效性、设备有效性、一致性、可追踪性、可见性、硬件环境无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、简单性、结构性
设计	精确性、健壮性、处理有效性、完备性、一致性、可追踪性、可见性、硬件环境无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、简单性、结构性
编码	精确性、健壮性、一致性、可追踪性、硬件环境无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、简单性、结构性
单元测试	精确性、健壮性、处理有效性、可操作性、通信有效性、完备性、一致性、可追踪性、可见性、模块性、清晰性、简单性、结构性
验收	完备性、精确性、健壮性、处理有效性、设备有效性、可操作性、培训性、一致性、可追踪性、文档完备性
维护	精确性、健壮性、设备有效性、可操作性、培训性、一致性、可追踪性、可见性、可扩充性、清晰性、文档完备性

作为独立第三方软件测试组织对软件进行静态测试，并对软件产品质量进行评测或对软件产品质量进行度量和评估的依据，就是前面叙述的软件质量框架“质量特征—质量子特征—度量元”三层质量模型。

在软件开发中，软件度量的根本目的是管理的需要。没有软件过程的可见度就无法对软件进行管理，没有软件产品质量的定量描述就无法对软件质量进行评价。度量是一种可用于决策的可比较的对象。软件度量包含费用、工作量、生产率、性能、可靠性和质量等方面的度

量。对于软件质量度量应根据软件质量要求来确定各个质量特性要求的级别(评定等级),标识每个质量子特性所要求的度量元和度量方法(事实上,软件质量特性和子特性描述的软件度量需求很难直接测量,需要进一步确定相关的度量元,并将它们与质量子特性、质量特性以及质量模型联系起来)。最终确定软件产品质量的定量定级水平。

软件产品质量评价准则是用来确定一个特定软件产品的总体质量是否能够被接受的已经定义成文的规则和条件的集合。

1. 软件质量的度量过程

软件质量度量就是从整体上对软件质量进行测评,用于软件开发中对软件进行质量控制,并最终对软件产品进行评价和验收。

IEEE Std 1061 软件质量度量方法学提供了系统地进行软件质量度量的途径,跨越整个软件生命周期,并包括下列 5 个步骤。

(1) 建立软件质量需求。质量需求表达了在具体应用的特定环境下对软件产品质量的定量要求,应该在软件开发前或初期进行定义,它是有效构造软件质量和客观评价质量的前提。质量需求规格说明可通过所需质量特性的直接度量及其直接度量目标值进行定量表示。直接度量用来验证最终产品是否达到了质量需求。

(2) 准备度量。由软件质量特性和子特性描述的软件质量需求常常无法直接测量,需要进一步确定相关的度量元。在度量的准备阶段,应根据应用环境,为软件开发的各个阶段和其最终产品分别确定适当的度量元,建立度量元、质量子特性、质量特性的映射模型,确定合理的评估准则。

(3) 实现软件质量度量。数据收集过程规定从数据收集点到度量评价的数据流程,确定有关数据的收集条件,给出工具的使用说明及数据存放规程。在全面实施度量前,最好首先在小范围内试验数据收集和度量计算规程,分析其数据量是否一致、度量要求是否确切,尤其要检查主观判断的数据说明和要求是否清晰;然后检查样板度量过程的费用,修改或完善费用分析;最后检查所收集到的数据的准确性、度量单位的合适性、所收集到的数据之间的一致性,确认数据样本的随机性、最小样本数、相似性等。

(4) 分析质量度量结果。分析并报告度量结果不仅要做出度量和评估的结论,还要进行度量元的确认,从而确定哪些度量元的确适用于当前软件质量度量活动并可以用于预测软件质量特性值,根据这些度量值和由此计算得到的直接度量的预测值决定被度量对象是否需要做进一步的度量和分析。

(5) 确认软件质量度量。把预测的度量结果与直接度量结果进行比较,以确定预测的度量是否准确地测定了它们的相关质量要素。

2. 度量元选择原则

在对软件质量特性、子特性进行度量时,要对度量元进行适用性选择,其选择原则是:

① 选择充分体现该领域软件特征的度量元。

② 可操作性好、度量元数据易获得且其获取的代价较小。

③ 少而精,规模适中。

④ 子特性、度量元尽量不相关。

⑤ 标准符合性要突出。

在选择度量元并进行实际运用时,我们一定要避免走入软件度量的误区。例如:

(1) 目的不明,事后发现度量的内容与管理无关。

(2) 使用度量去评价个人。

（3）开发人员拒绝执行，认为会影响其工作业绩。

（4）度量过多，要求广泛收集数据，程序烦琐，不堪重负。

（5）认为度量结果报告无法引导管理活动。

（6）管理部门看到可能发生的问题或无成功的结果，而放弃支持度量工作。

（7）过分强调单个因素的度量。

3. 软件质量评价指标（评价准则）的确定

针对具体软件产品或软件项目实施度量评价时，要确定评价指标。也就是说衡量一个软件产品或中间产品的好坏，质量特性、子特性及度量元的合格与否要给出准绳，给出每个特性、子特性的权重。这样一些数据就需要长期积累、总结，也包括专家的评估确定。

因此，选择合适的软件质量指标体系并使其量化是软件测试与评估的关键。评估指标可以分为定性指标和定量指标两种。理论上讲，为了能够科学客观地反映软件的质量特征，应该尽量选择定量指标。但并不是所有的质量特征都可用定量指标进行描述，有时要采用一定的定性指标。这样，我们在选取评估指标时，可按如下原则来进行。

（1）针对性：不同于一般软件系统，能够反映评估软件的本质特征，具体表现就是功能性与高可靠性。

（2）可测性：可定量表示，可通过数学计算、平台测试、经验统计等方法得到具体数据。

（3）简明性：易于被各方理解和接受。

（4）完备性：选择的指标应覆盖分析目标所涉及的范围。

（5）客观性：客观反映软件本质特征，不能因人而异。

另外，我们要注意的是选择的评估指标不是越多越好，关键在于指标在评估中所起的作用。评估时指标太多，会增加结果的复杂性，甚至还会影响评估的客观性。指标的确定一般是采用自顶向下，逐层分解，并在动态过程中反复综合平衡。

4. 软件质量特性的评价策略

1）功能性指标

功能性是最重要的软件质量特征之一，可以细化成完备性和正确性，目前对软件的功能性评价主要采用定性评价方法。在这里，完备性是与软件功能完整、齐全有关的软件属性。软件实际完成的功能少于或不符合需求规定的那些功能，则不能说该软件的功能是完备的；正确性是与能否得到正确或相符的结果或效果有关的软件属性。软件的正确性很大程度上与软件模块的工程模型和软件编程人员的水平有关。对这两个子特性的评价依据主要是软件功能性测试的结果，评价标准则是软件实际运行中所表现的功能与规定功能的符合程度。

在软件的需求中，明确规定了该软件应该完成的功能。准备进行验收测试的软件应该具备这些明确或隐含的功能。

对于软件的功能性测试主要针对每种功能设计若干典型测试用例。软件测试过程中运行测试用例，然后将得到的结果与已知标准答案进行比较。测试用例集的全面性、典型性和权威性是功能性评价的关键。

2）可靠性指标

按软件评测要求，可靠性可细化为成熟性、稳定性、易恢复性等。对于软件的可靠性评价主要采用定量评价方法：选择合适的可靠性度量元，然后分析可靠性数据而得到参数具体值，最后进行评价。经过对软件可靠性细化分解并参照需求，可以得到软件的可靠性度量元。

3）可用性指标

可用性可以细化为易理解性、易学习性和易操作性等。这三个子特性主要是针对用户而

言的,对软件的可用性评价主要采用定性评价方法。

易理解性是与用户认识软件的逻辑概念及其应用范围所花的努力有关的软件属性,要求软件研制过程中形成的所有文档语言简练、前后一致、易于理解以及语句无歧义;易学习性是与用户为学习软件应用(例如运行控制、输入、输出)所花的努力有关的软件属性,要求研制方提供的文档(如《计算机系统操作员手册》、《软件用户手册》和《软件程序员手册》)内容详细、结构清晰、语言准确;易操作性是与用户为操作和运行控制所花的努力有关的软件属性,要求软件的人机界面友好、界面设计科学合理以及操作简单等。

4) 效率指标

效率可以细化成时间特性和资源特性,对软件的效率评价采用定量方法。

效率度量内容参见图 5-13。图中的度量元属软件内部表现,需用专门的测试工具和特殊的途径才可获得,将结果与任务书中的指标进行比较,得到的结果可作为效率评价的依据。

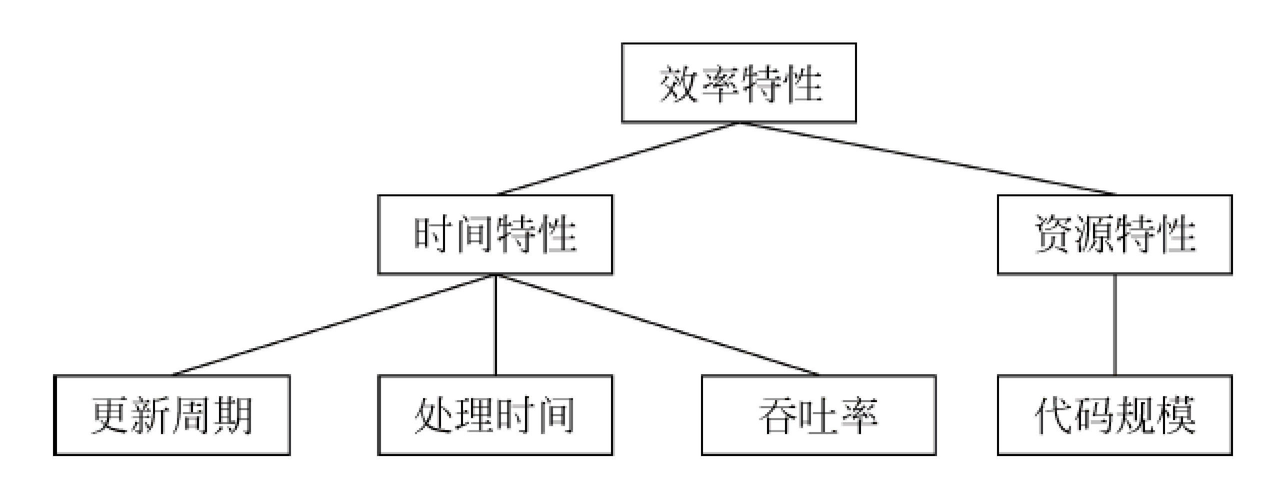


图 5-13 效率度量参数结构图

5. 软件质量定量评价公式

对于软件质量的定量评价,国内外在这方面做了很多研究工作,取得了一定的成果。国外著名软件质量度量和评价产品中都给出了相关的计算公式,如 Panorama++,Logiscope,McCabe IQ 等。下面我们结合这些公司的软件质量定量评价公式进行计算公式的介绍。

(1) 可维护性:可维护性指当系统的功能发生变化和升级时或当发现错误时容易修改的特性,一个可维护的软件应该是可理解的和可测试的,因为只有这样软件人员才能够容易地确定其影响并验证其变化。

可维护性=0.5×可测试性+0.5×可理解性

(2) 可测试性:可测试性指容易验证软件的正确功能,影响一个程序可测试性的两个软件特性是结构性和复杂性。为了正确地操作,高度结构化的程序容易把系统部件分成几个独立的测试部分,每部分的复杂性对正确地进行测试所需的测试程序量有着影响。

可测试性=0.5×结构性+0.5×McCabe 复杂度

(3) 可理解性:可理解性指不是原设计者/程序员的那些人员能容易理解一个程序的功能的程度。它要求简化程序,程序应有描述性注释、良好的结构、最小的复杂性,程序编写要求简明。

可理解性=0.25×结构性+0.25×McCabe 复杂度+0.25×简洁性+0.25×自描述性

(4) 结构性:结构性指程序自身的特性,它测量一个程序结构的好坏。衡量一个程序结构是否良好有下列 5 个方面:

① 应不修改全局数据。

② 对逻辑嵌套的深度要有限制。

③ 有单一的返回,不使用 goto 语句。

④ 使用/设置全部参数类型和对象。

⑤ 推荐使用简单的循环语句而不是 while 循环语句。

结构性＝0.2×编码语句的最大嵌套层次＋0.2×修改全局数据＋0.2×使用 goto 语句＋
0.2×数据习惯用法＋0.2×无条件循环语句所占比例

(5) 复杂性：复杂性反映全部程序和它的部件的复杂状态。对于程序单元(过程或函数)，一般采用 McCabe 圈复杂度计算复杂性，分析整个编码的执行控制；对于应用程序，程序单元数量和它们之间的相互关系影响复杂性。

软件复杂性＝所有模块复杂性/所有模块

模块复杂性＝(圈复杂度＋模块设计复杂度＋设计复杂度＋集成复杂度)/(圈复杂度临界值＋
模块设计复杂度临界值＋设计复杂度临界值＋集成复杂度临界值)

(6) 简洁性：简洁性指多余信息不在程序中出现的程度。

简洁性＝0.4×实体的习惯用法＋0.4×局部调用＋0.2×被调用

其中：

实体的习惯用法＝{[1－(未使用非输出对象声明/对象声明)]＋
[1－(未使用非输出类型声明/类型声明)]＋
[1－(未使用非输出参数声明/参数声明)]}/3

局部调用是在同一个封闭的父单元内对其他程序单元的调用。

(7) 自描述性：自描述性是衡量一个程序如何详细地描述自己，自动检查是否存在特殊类型的注释，以判断程序本身描述的质量。

自描述性＝0.2×空格行所占比例＋0.3×全部注释行所占的比例＋0.5×注释实体所占比例

(8) 可移植性：可移植性指在一种平台上开发的程序能容易地移植到另一种平台上的程度，使得系统的改变对操作不会有不利的影响。

可移植性＝0.5×独立性＋0.5×完整性

(9) 独立性：独立性表示一个程序与开发环境或主机环境脱离的程度。

独立性＝0.5×异常比例＋0.5×用户定义类型

(10) 完整性：完整性指没有信息遗漏，测量一个程序被完成的程度。

完整性 ＝(if 语句＋case 语句＋初始化对象)/3

(11) 可靠性：可靠性指测量一个程序正确操作的置信度，软件的缺陷越少可靠性越高。

可靠性＝0.33×完整性＋0.33×模块性＋0.34×可测试性

(12) 模块性：模块性指一个程序按它的功能分割成几个独立的程序单元的程度。独立实现程序功能的那些程序具有高度的模块性。

模块性＝0.5×编码行数＋0.5×结构性

6. 源程序的度量

一般高级语言的源程序度量都要有一些基本的度量元来支持，如 McCabe 圈复杂度、注释率、嵌套层数、执行路径个数、宏定义数、函数参数个数、指令个数、goto 数、return 数、扇入扇出数等。而面向对象语言有关面向对象属性的度量有每个类的含权方法数、类的可测试性、子孙数、祖先数、继承数深度、对象间耦合、多继承表示、类的扇入扇出数、类的注释率、类耦合、重定义方法数等。图 5-14 是被测程序度量过程的几个示意图。图 5-15 是软件质量度量的饼图和 KV 图(雷达图或蜘蛛图)。

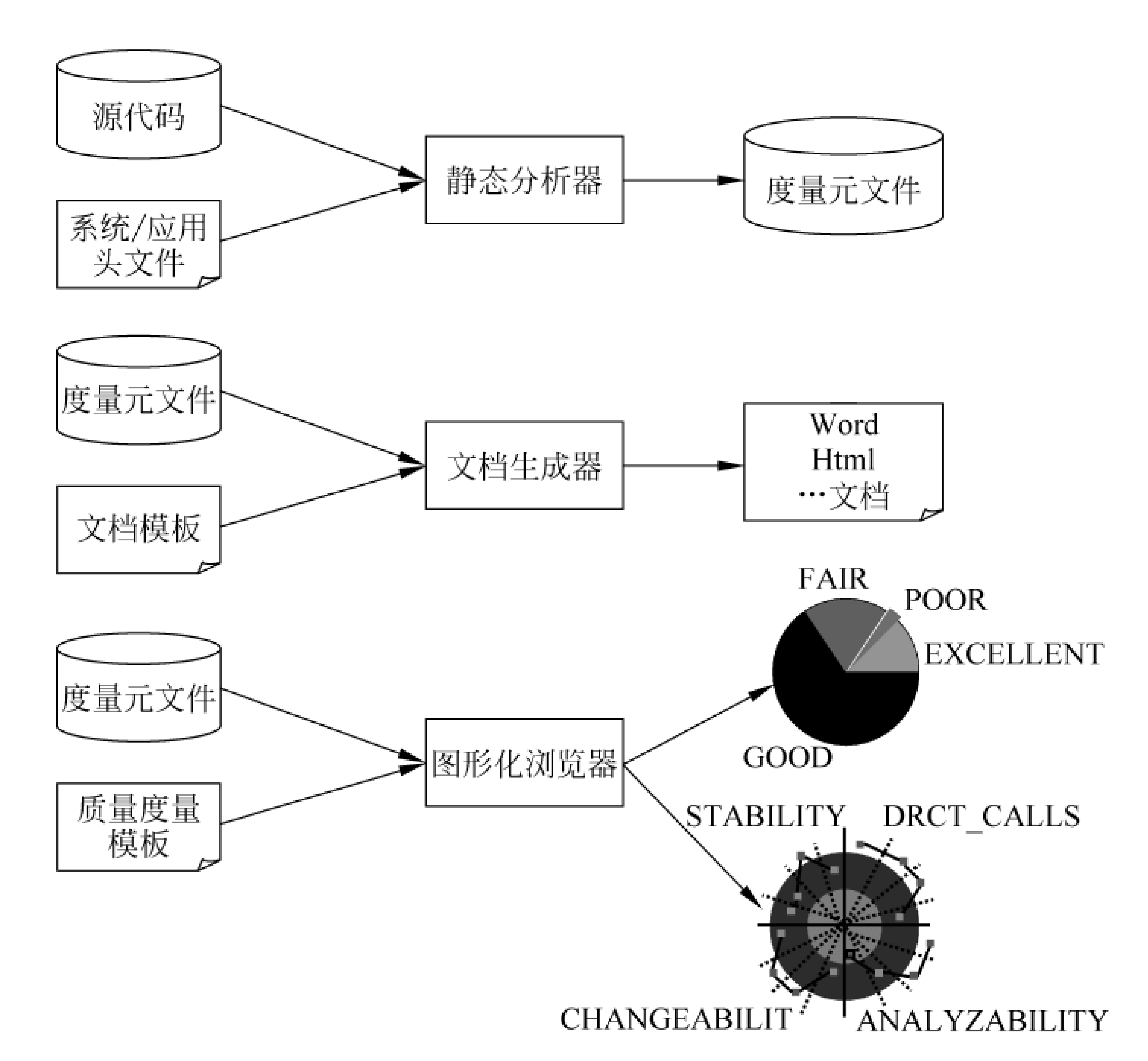


图 5-14 被测程序度量过程示意

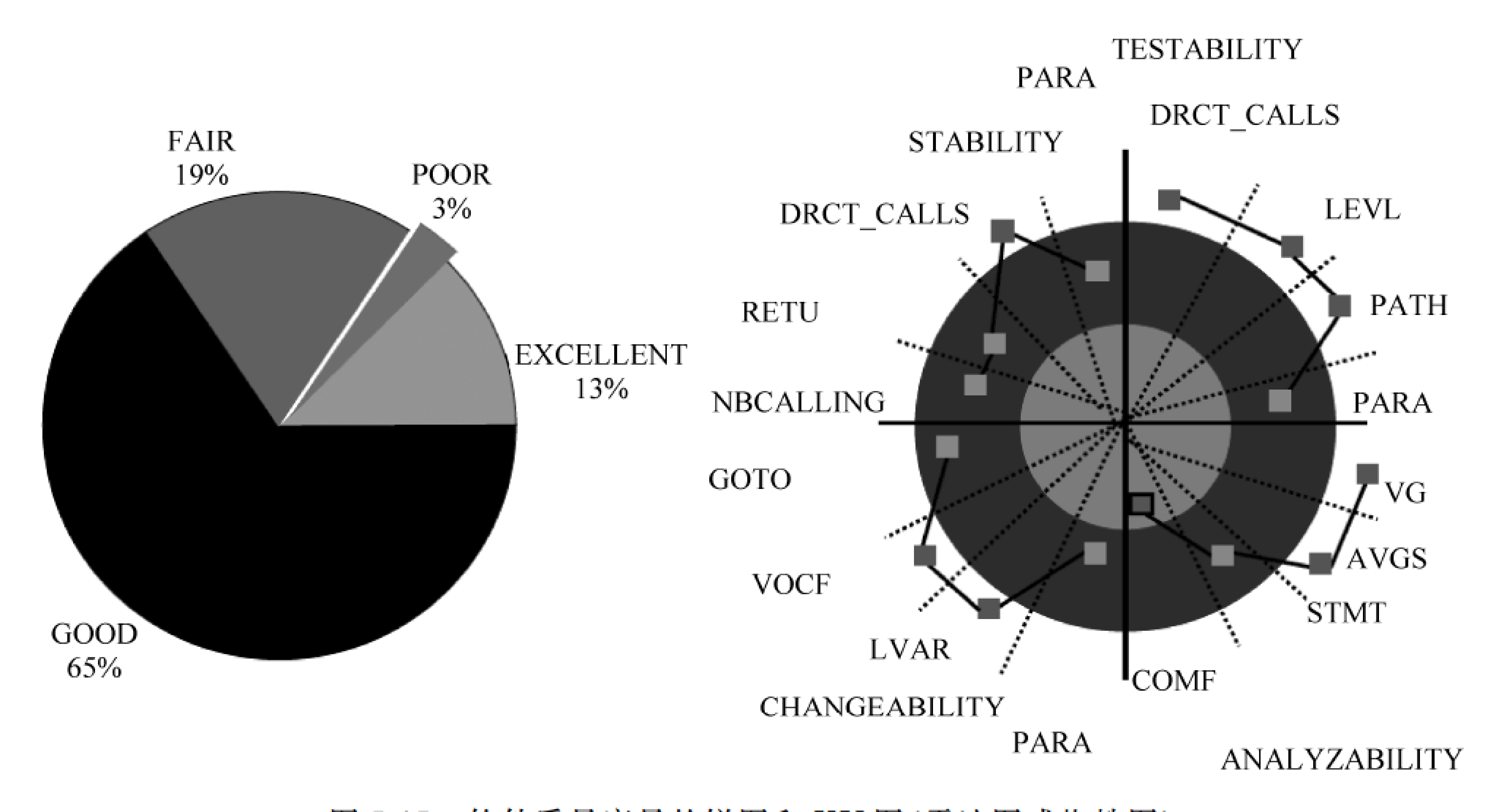


图 5-15 软件质量度量的饼图和 KV 图(雷达图或蜘蛛图)

5.5 静态分析工具

静态分析程序不需要执行被测程序,它扫描被测程序的正文,对程序的数据流和控制流进行分析。然后输出测试报告。通常,它具有以下几类功能。

(1) 对模块中的所有变量,检查是否都已定义,是否引用了未定义的变量,是否有已赋过值但从未使用的变量。实现方法是建立变量的交叉引用表。

（2）检查模块接口的一致性。主要检查子程序调用时形式参数与实际参数的个数、类型是否一致，输入、输出参数的定义/使用是否匹配，数组参数的维数，下标变量的范围是否正确，各子程序中使用的公用区（或外部变量、全局变量）定义是否一致，等等。

（3）检查在逻辑上可能有错误的结构以及多余的不可达的程序段。如交叉转入、转出的循环语句，为循环控制变量赋值，存取其他模块的局部数据等。

（4）建立“变量/语句交叉引用表”、“子程序调用顺序表”、“公用区/子程序交叉引用表”等。利用它们找出变量错误可能影响到哪些语句，影响到哪些其他变量等。

（5）检查被测程序违反编程标准的错误。例如模块大小、模块结构、注释的约定、某些语句形式的使用，以及文档编制的约定等。

（6）静态特性的统计功能：各种类型源语句的出现次数，标识符使用的交叉索引，标识符在每个语句中使用的情况，函数与过程引用情况，任何输入数据都执行不到的孤立代码段，未经定义的或未曾使用过的变量，违背编码标准之处，公共变量与局部变量的各种统计。

静态分析工具一般由四部分组成：语言程序的预处理器、数据库、错误分析器和报告生成器。预处理器把词法分析与语法分析结合在一起，以识别各种类型的语句。源程序被划分为若干程序模块单元（如主程序与一些子程序），同时生成包含变量使用、变量类型、标号与控制流等信息的许多表格。有些表格是全局表，它们反映整个程序的全局变量信息，如模块名、函数及过程调用关系、全局变量等。有些表格是局部表，它们对应到各个模块，记录模块中的各种结构信息，如标号引用表、分支索引表、变量属性表、语句变量引用、数据或记录特性表等。

由于静态分析工具使用算法技术检查源代码中的错误，标明问题区域，帮助编程人员做更详细的检查。这种算法方式无须使用测试用例，算法本身决定了分析工具发现错误的效率。近十几年来，随着研究人员开发出更加高效的算法，静态分析工具变得越来越强大。

5.5.1　IBM Rational Logiscope RuleCheck/Audit 介绍

Logiscope 原为法国 Telelogic 公司专用于软件质量保证和软件测试的产品，现已被 IBM 收购。其主要功能是对软件做质量分析和测试以保证软件的质量，并可做认证、逆向工程和维护，特别是针对要求高可靠性和高安全性的软件项目和工程。

Logiscope 应用于软件的整个生命周期，它贯穿于软件需求分析阶段→设计阶段→代码开发阶段→软件测试阶段（代码审查、单元/集成测试和系统测试）→软件维护阶段的质量验证要求。

在设计和开发阶段，使用 Logiscope 可以对软件的体系结构和编码进行确认。可以在尽可能的早期阶段检测那些关键部分，寻找潜在的错误，并在禁止更改和维护工作之前做更多的工作。在构造软件的同时，就定义测试策略。可帮助编制符合企业标准的文档，改进不同开发组之间的交流；在测试阶段用 Logiscope，使测试更加有效。可针对软件结构，度量测试覆盖的完整性，评估测试效率，确保满足要求的测试等级。特别是，Logiscope 还可以自动生成相应的测试分析报告；在软件的维护阶段，用 Logiscope 验证已有的软件是否质量已得到保证，对于状态不确定的软件，Logiscope 可以迅速提交软件质量的评估报告，大幅度地减少理解性工作，避免非受控修改引发的错误。

Logiscope 的最终目的是评估和提高软件的质量等级，采用基于国际间的标准度量方法（如 Halstead、McCabe 等）的质量模型对软件进行分析，从软件的编程规则、静态特征和动态测试覆盖等多个方面，量化地定义质量模型，并检查、评估软件质量。

（1）Logiscope 能够分析和计算出 ISO/IEC 9126 定义的质量特性。

（2）Logiscope 为 ISO 9001 提供需求（Test Acceptance Criteria And Quality Records），自

动确定 ISO 9001 认证过程的任务。

(3) Logiscope 提供 SEI/CMM 在第 2 级(Repeatable)所要求的软件质量跟踪等关键实践的要求,推进开发组织尽快达到 SEI/CMM 的第 3 级。

(4) 在有合同关系时,合同方可以用 Logiscope 明确定义验收时质量等级和执行测试。承制方可以用 Logiscope 验证其软件的质量。

(5) 对所有开发者提供确保代码质量和进行有效测试的方法。

(6) 创立公司的技术文化。

(7) 对项目管理者和质量工程师提供对整个项目进行制度化的测试和评估。

Logiscope 有三项主要功能,以三个独立工具的形式出现,分别是:①软件质量分析工具 Audit,主要用来静态检查程序代码的各种指标,如健壮性、可维护性等;②代码规范性检测工具 Rulechecker,主要用来静态检查程序代码是否符合相应的编码规范;③测试覆盖率统计工具 TestChecker,主要用来动态测试程序代码。其中 Audit 和 Rulechecker 提供了对软件进行静态分析的功能,TestChecker 提供了测试统计的功能。

Logiscope 可以测试 C、C++、Ada 和 Java 的程序,而且具有跨平台的特性,可以运行在 Windows 和 UNIX 等各种平台上。

1. 软件质量分析工具 Audit

关于软件质量模型的相关知识,已经在 5.4 节讲过,这里不再介绍了。

Audit 以 ISO 9126 模型作为软件质量评价模型的基础。软件质量评价模型描述了从 Halstend、McCabe 的度量方法学引入的质量方法学中的质量因素(可维护性、可重用性等)、质量准则(可测试性、可读性等)和质量度量元。即本模型是一个三层的机构组织:质量因素、质量准则和质量度量元。

质量因素是从用户角度出发,对软件的质量特性进行总体评价;质量准则是从软件设计者角度出发,设计为保障质量因素所必须遵循的法则;质量度量元从软件测试者角度出发,验证是否遵循质量准则。一个质量因素由一组质量准则来评估;一个质量准则由一组质量度量元来验证,其关系如图 5-16 所示。

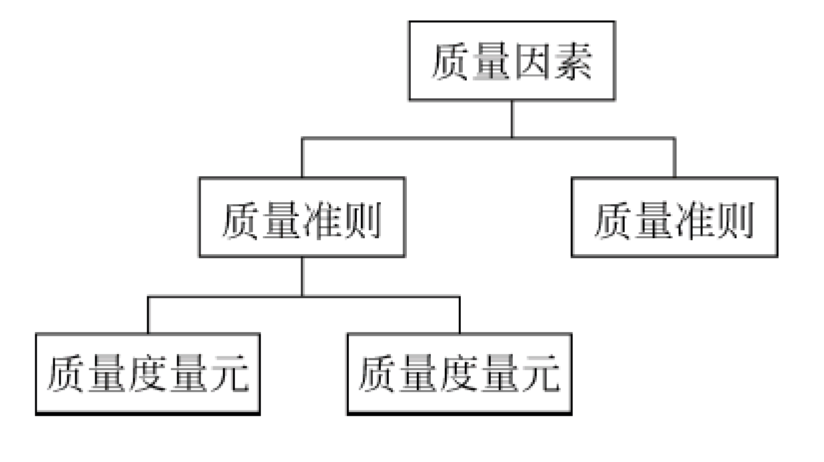


图 5-16 Logiscope 定义的质量结构关系

Logiscope 从系统、类和函数三个层次详细规定了上述质量特性及其组成关系。下面以 C++ 程序的类层为例:

(1) 质量因素(如可维护性、可重用性等,如图 5-17 所示);



图 5-17 Logiscope 定义的质量因素

(2) 质量准则(如可分析性、可修改性、稳定性、可测试性等,如图 5-18 所示);

(3) 质量度量元(由于质量度量元比较众多,故在此不详细描述)。

Audit 通过一个文本文件来定义质量模型,在文件中首先定义了若干个质量元,并为这些度量元设定了数值范围,接着通过组合若干个度量元形成质量标准,最后又通过组合质量标准,形成最后的质量因素。而这个过程与软件质量模型中由低层到高层、由细节到概括的结构

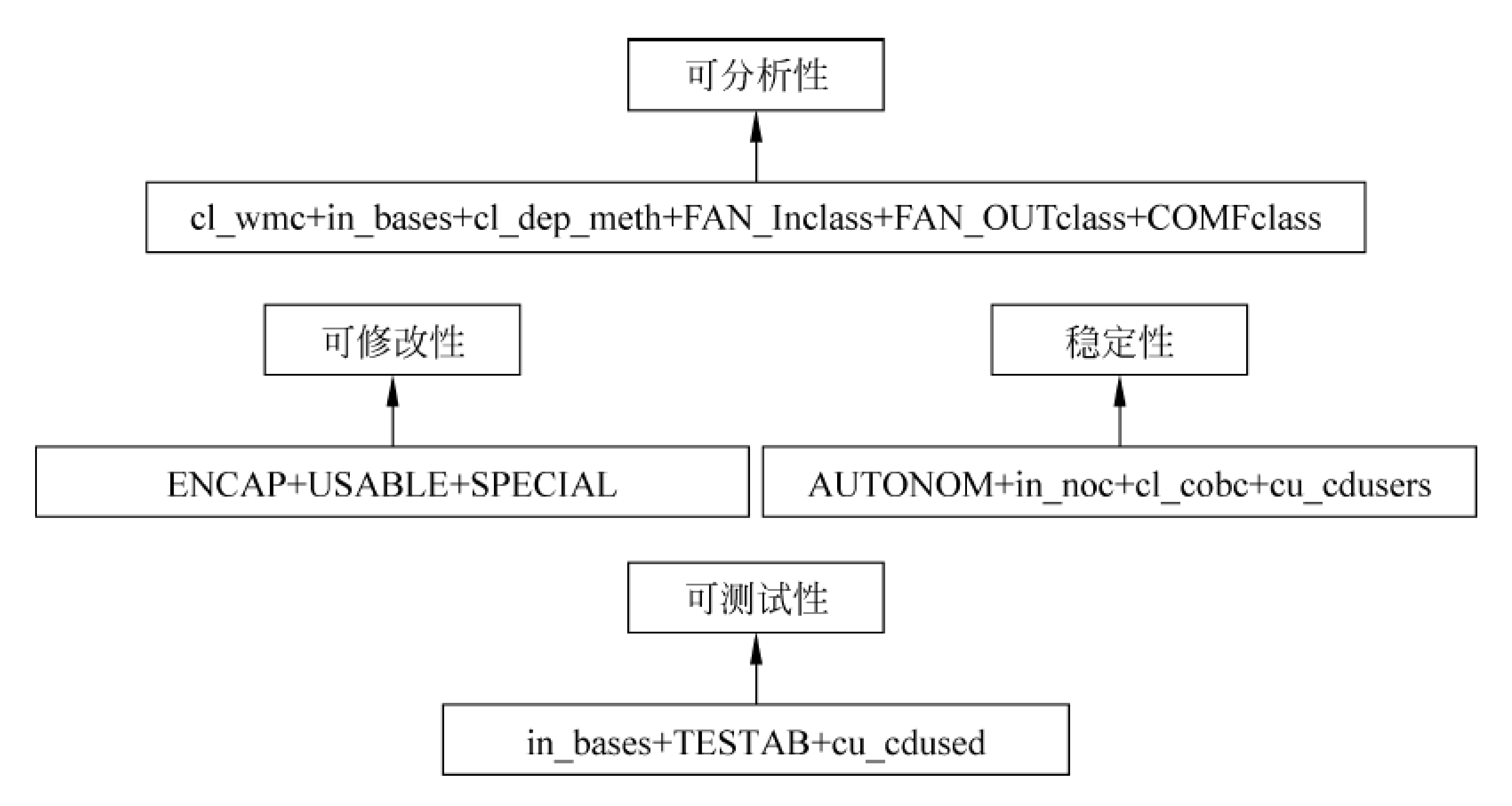


图 5-18　Logiscope 定义的质量准则

恰好对应。然后 Audit 将被评价的软件与所选的质量模型进行比较,生成软件质量分析报告,并用图形的形式显示软件质量的等级,因此,质量人员可以把精力集中到需要修改的代码部分,可以对度量元素和质量模型不一致的地方作出解释并提出纠正的方法。通过对软件质量进行评估及生成控制流图和调用图发现最大可能发生错误的部分。一旦发现这些部分,可以使用度量元及控制流图、调用图等手段做进一步分析。

Audit 可以生成如下分析结果,并以图形的方式显示出来。

(1) 质量报告。Logiscope 根据质量模型,生成相应的软件质量分析报告(HTML 形式),如图 5-19 所示。

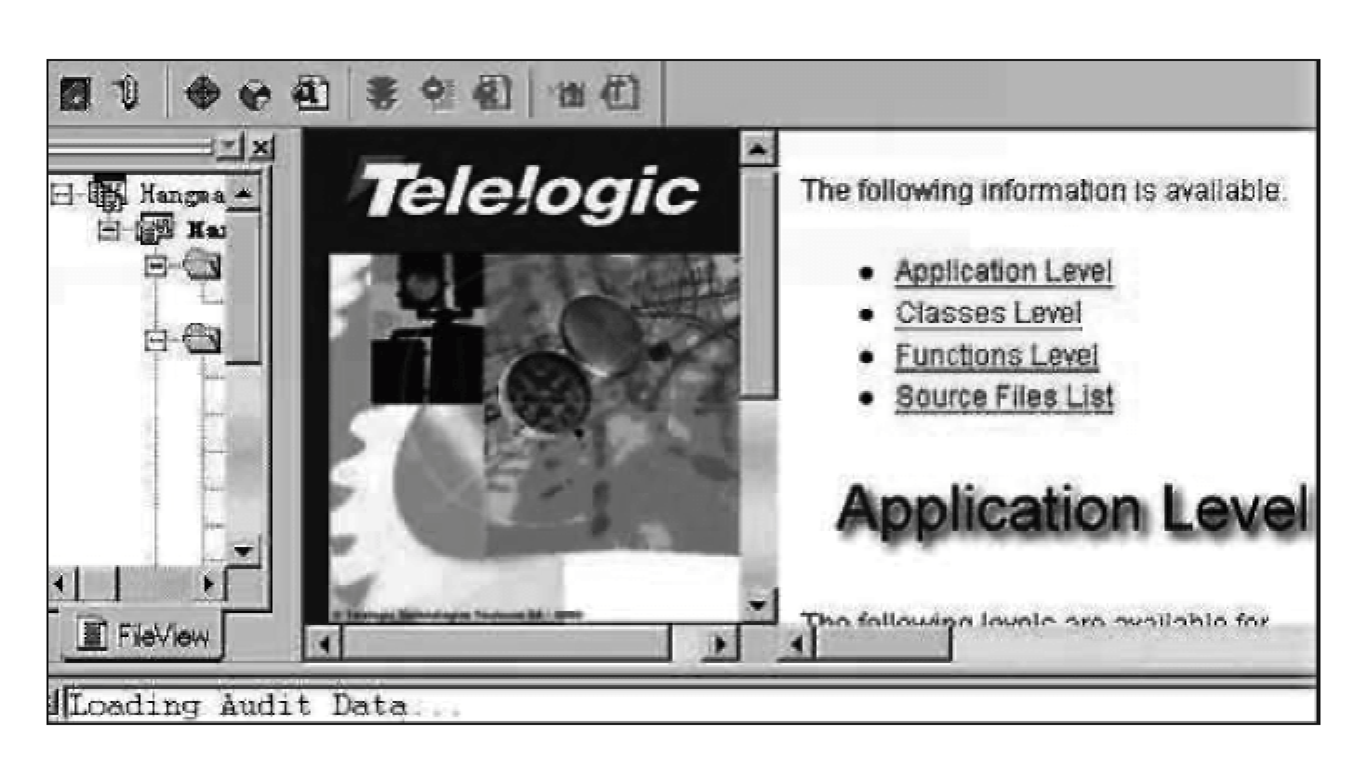


图 5-19　Logiscope 生成的软件质量分析报告

(2) 质量度量元。通过图形的方式可以清楚地分析和观察每个类或方法中的质量度量元的数值,并判断其是否合法,如图 5-20 所示。

Metric : CString::GetAt	Value	Max	Min	Status
AVGS: Average size of statements	-1.00	9.00	1.00	-1
COMF: Comments frequency	-1.00	+oo	0.20	-1
FAN_IN: Fan In	-1.00	4.00	0.00	-1
FAN_OUT: Fan Out	-1.00	4.00	0.00	-1
LEVL: Number of levels	-1.00	4.00	1.00	-1

Messages　Build　Browse

图 5-20　质量度量元列表清单

(3) 质量准则。可以清楚地分析和判断各质量因素所含有的质量准则的数据和合格性，如图 5-21 所示。

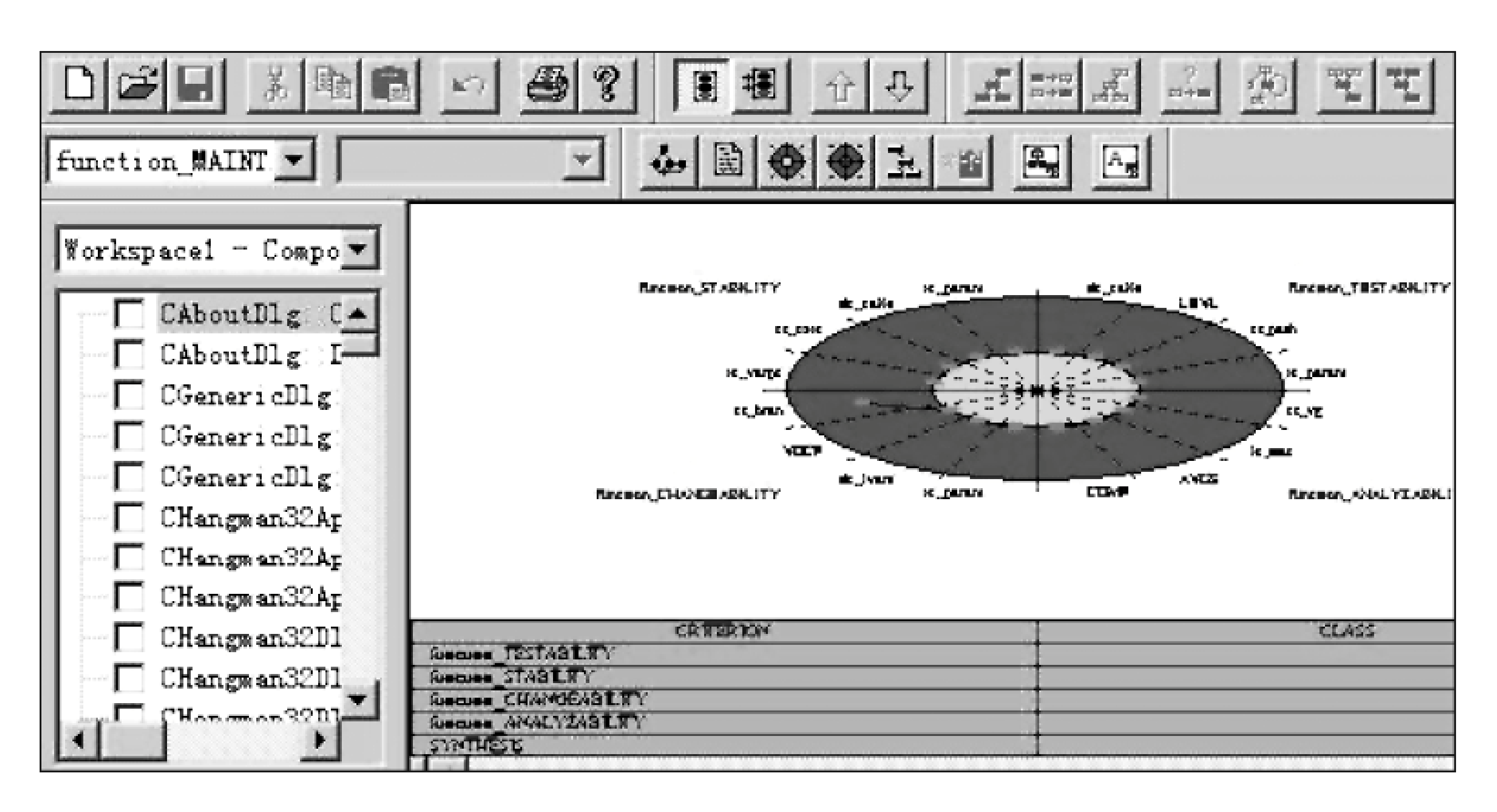


图 5-21　基于 Kiviat 图质量准则

(4) 质量因素图。针对系统层、类层和函数层，分别分析质量因素的合格性和所占百分比，如图 5-22 所示。

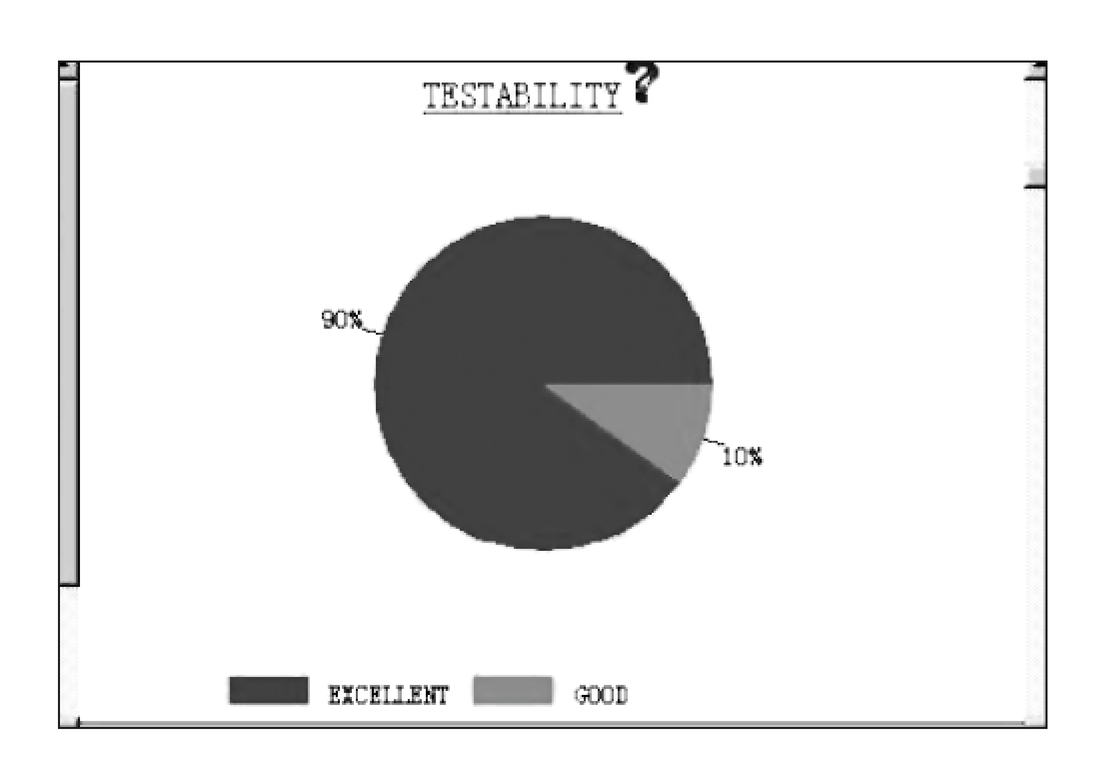


图 5-22　基于饼图的质量因素

(5) 程序流程图。控制流图显示算法的逻辑路径。其图形表示适用于评价函数的复杂性，如图 5-23 所示。

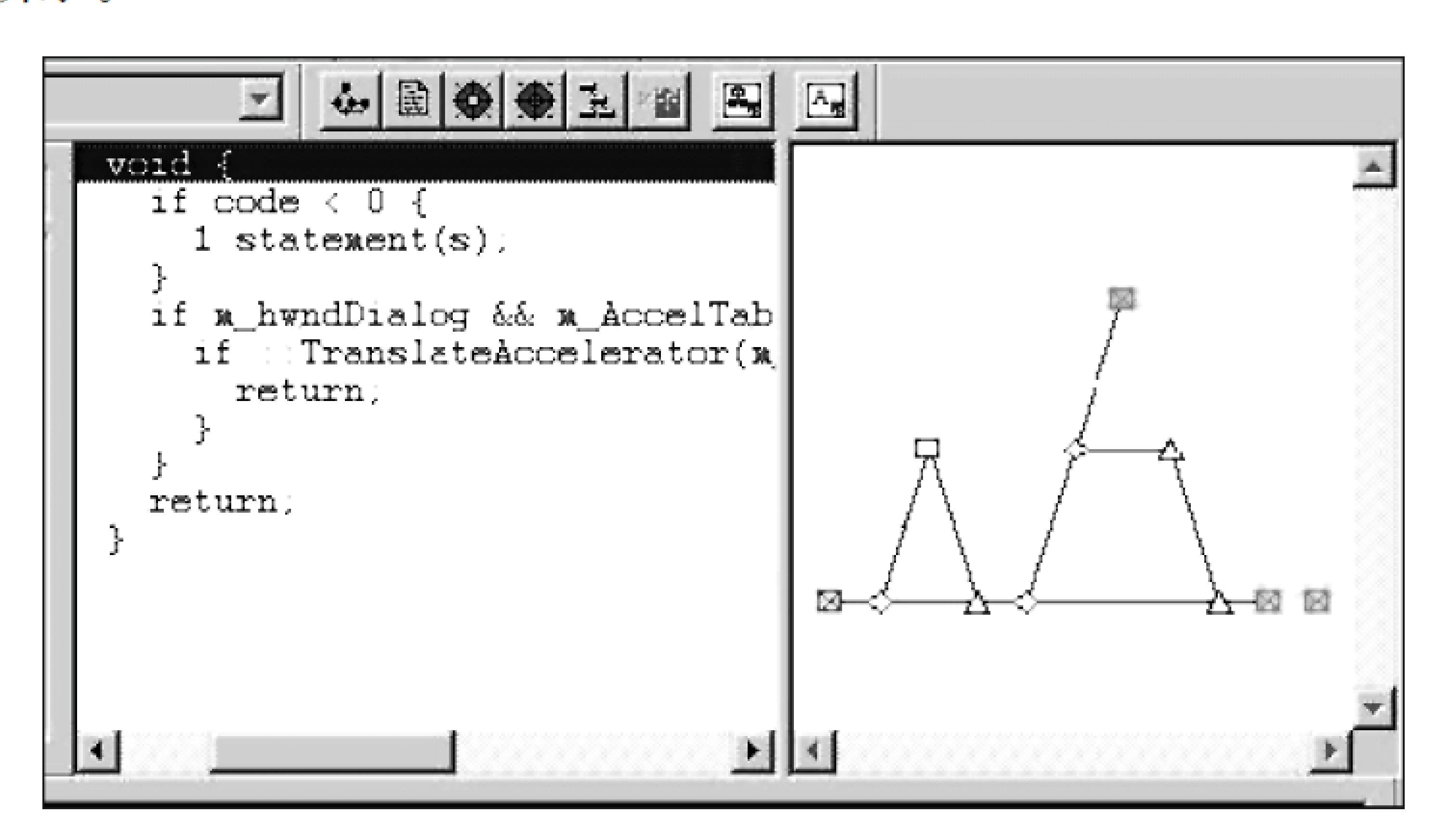


图 5-23　程序流程图

（6）程序调用图。调用图显示过程和函数之间的关系，非常适用于检查应用系统的设计，如图 5-24 所示。

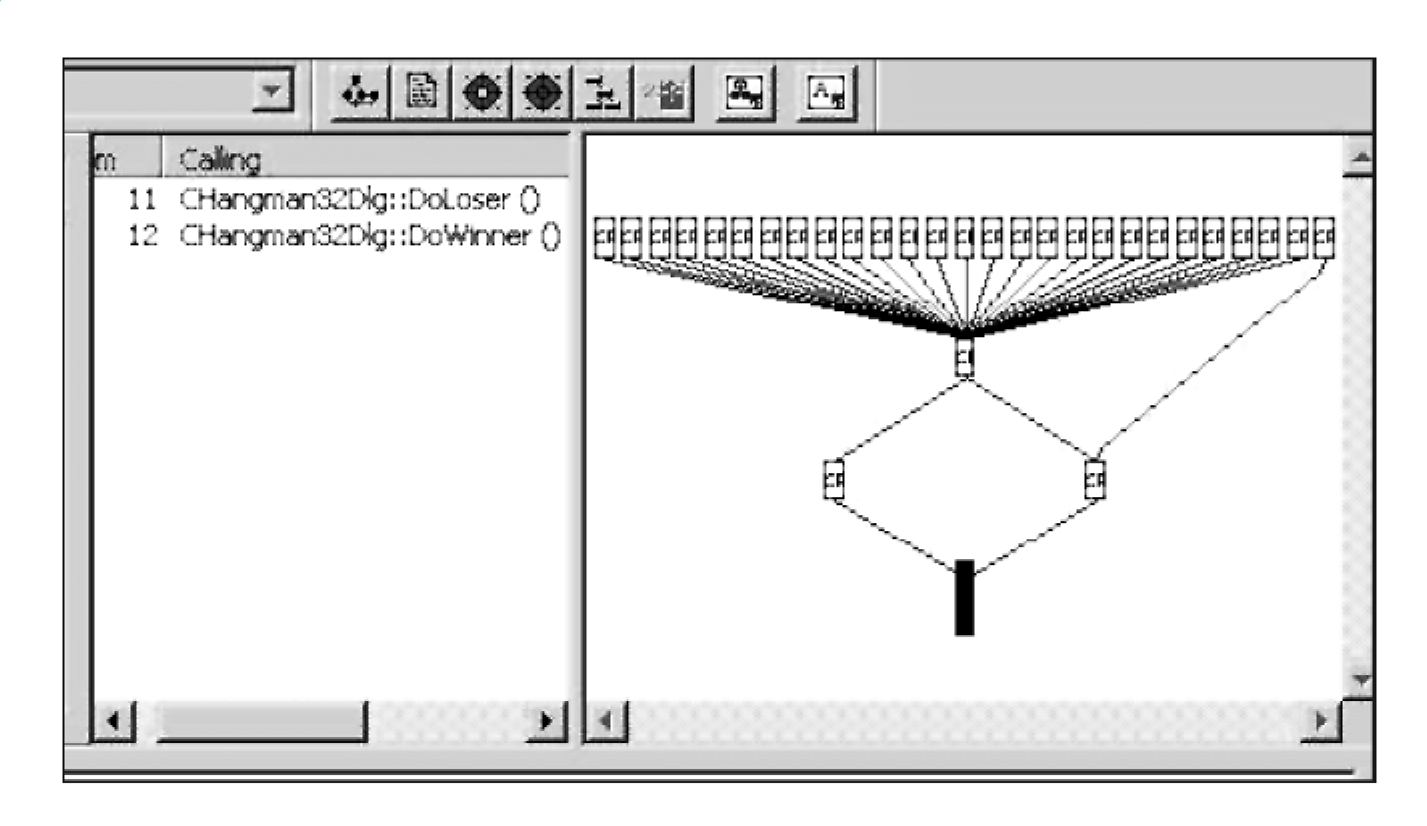


图 5-24　程序调用图

（7）Kiviat 图。使质量等级与所选择的参考之间的一致性对比更加可视化，如图 5-25 所示。

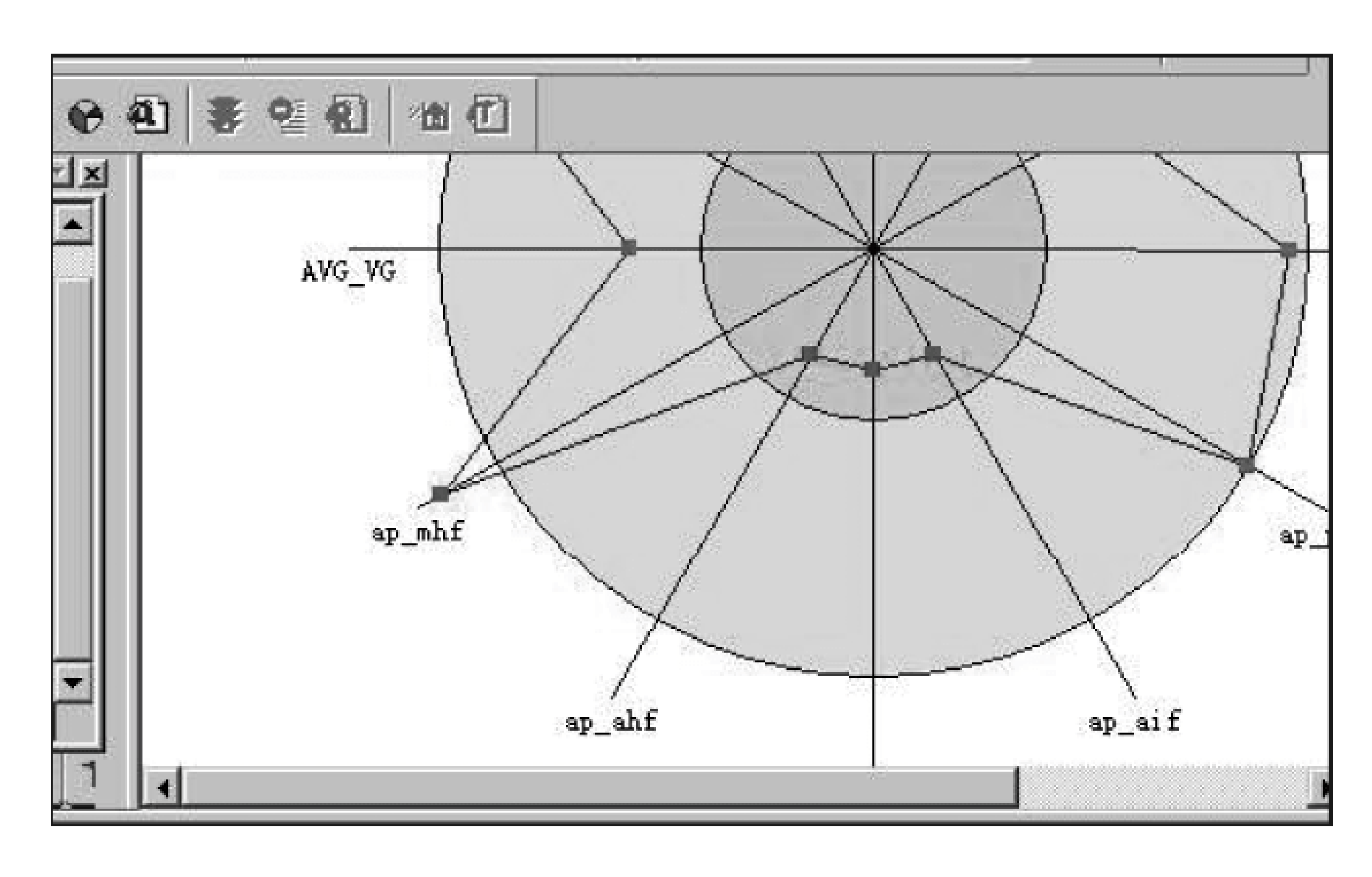


图 5-25　Kiviat 图

2. 代码规范性检测工具——Rulechecker

Rulechecker 工具是为了协助程序员实现代码更健壮、可读性更好的目的。因此，Logiscope 提供编码规则与命名检验，这些规则根据业界标准和经验所制定。因此可建立企业可共同遵循的规则与标准，而避免自我不良的编程习惯及彼此不相容的困扰。

Rulechecker 预定义了 50 个编程规则：名称约定（例如局部变量用小写等）、表示约定（例如每一行一条指令）、限制（例如不能使用 goto 语句等）。这些编程规则形成了一个编码规范集。用户可以从这些规则中选择，用户可以对其内容进行定制，也可以使用 TCL 脚本和编程语言定义新的规则，这就大大增加了灵活性，使 Rulechecker 能更好地适应实际情况的需要。此外，还提供了 50 个面向安全关键系统的编程规则。

在为被测代码建立 Rulechecker 项目的过程中，有一步是 Choose a configuration file，这一步就是选择一个编码规范描述文件，Rulechecker 提供了一个叫“RuleChecker. cfg”编码规范描述文件，也可以修改或重新编写一个. cfg 文件来适用实际的要求。

5.5.2 HP FortifySCA 介绍

惠普于 2009 年 9 月完成了对互联网安全软件厂商 Fortify Software 的并购，使得惠普在数据应用安全分析领域占有领先地位，其技术可以防范恶意软件攻击，从而为用户提供一流的安全解决方案，保证用户从开发到运营整个环节的安全，帮助企业降低商业风险，而惠普自身也成为全球领先的软件安全产品解决方案供应商。

HP Fortify Software 软件产品主要包括业界最优秀的软件安全源代码扫描器、业界唯一的软件应用监控防卫器以及可以管理软件开发中的安全流程的管理平台等。

1. HP FortifySCA 产品组件介绍

HP FortifySCA 是 HP Fortify Software 产品最重要的组成部分，是一个静态的、白盒的软件源代码安全测试工具。它通过内置的 5 大主要分析引擎——数据流、语义、结构、控制流、配置流等对应用软件的源代码进行静态的分析。在分析的过程中，与它特有的软件安全漏洞规则集进行全面的匹配、查找，从而将源代码中存在的安全漏洞扫描出来，并给予整理报告。扫描的结果中不但包括详细的安全漏洞的信息，还会有相关的安全知识的说明及修复意见。

HP FortifySCA 是一个产品的套件，它是由内置的分析引擎、安全编码规则包、审查工作台、规则自定义编辑器和向导、IDE 插件 5 部分组件，帮助完成对源代码安全漏洞的扫描、分析、查看、审计等工作。简单介绍这 5 个部分如下。

(1) 分析引擎(内置 5 大分析引擎与规则包配合工作，从 5 个侧面全面地分析程序源代码中的安全漏洞)。

(2) 安全编码规则包(由多位顶级的软件安全专家，多年研究出来的数十万条软件安全漏洞特征的集合。目前能查找出来约 350 多种安全漏洞，内置在 SCA 中与分析引擎配合工作)。

(3) 审计工作台(一个用来查看、审计 SCA 分析出来的漏洞结果的综合的平台，它包含大量的丰富的软件漏洞的信息。它包括了漏洞的分级、漏洞产生的全过程、漏洞所在的源代码行数定位，以及漏洞的解释说明和推荐的修复建议等内容，极大地方便了用户对 SCA 的查看、审计等工作)。

(4) 规则自定义向导/编辑器(HP FortifySCA 的规则支持自定义功能，方便用户扩展 SCA 对漏洞的分析能力，所以 SCA 提供了一个用户自定义的向导和编辑器)。

(5) IDE 插件(为了方便用户使用 SCA 对程序源代码进行安全扫描，它提供了多种 IDE 工具的插件，如 Eclipse，Visual Studio，RAD，WSAD 等)。

2. HP FortifySCA 扫描引擎介绍

HP FortifySCA 主要包含五大分析引擎：

① 数据流引擎(跟踪、记录并分析程序中的数据传递过程所产生的安全问题)；

② 语义引擎(分析程序中不安全的函数、方法的使用的安全问题)；

③ 结构引擎(分析程序上下文环境、结构中的安全问题)；

④ 控制流引擎(分析程序特定时间、状态下执行操作指令的安全问题)；

⑤ 配置引擎(分析项目配置文件中的敏感信息和配置缺失的安全问题)。

另外，还可通过特有的 X-Tier™ 跟踪器跨越项目的上下层次、贯穿程序来综合分析问题。

3. HP FortifySCA 工作原理

图 5-26 所示为 HP FortifySCA 工作原理。

4. HP FortifySCA 扫描结果

HP FortifySCA 的扫描结果文件为.FPR 文件，包括详细的漏洞信息：漏洞分类、漏洞产

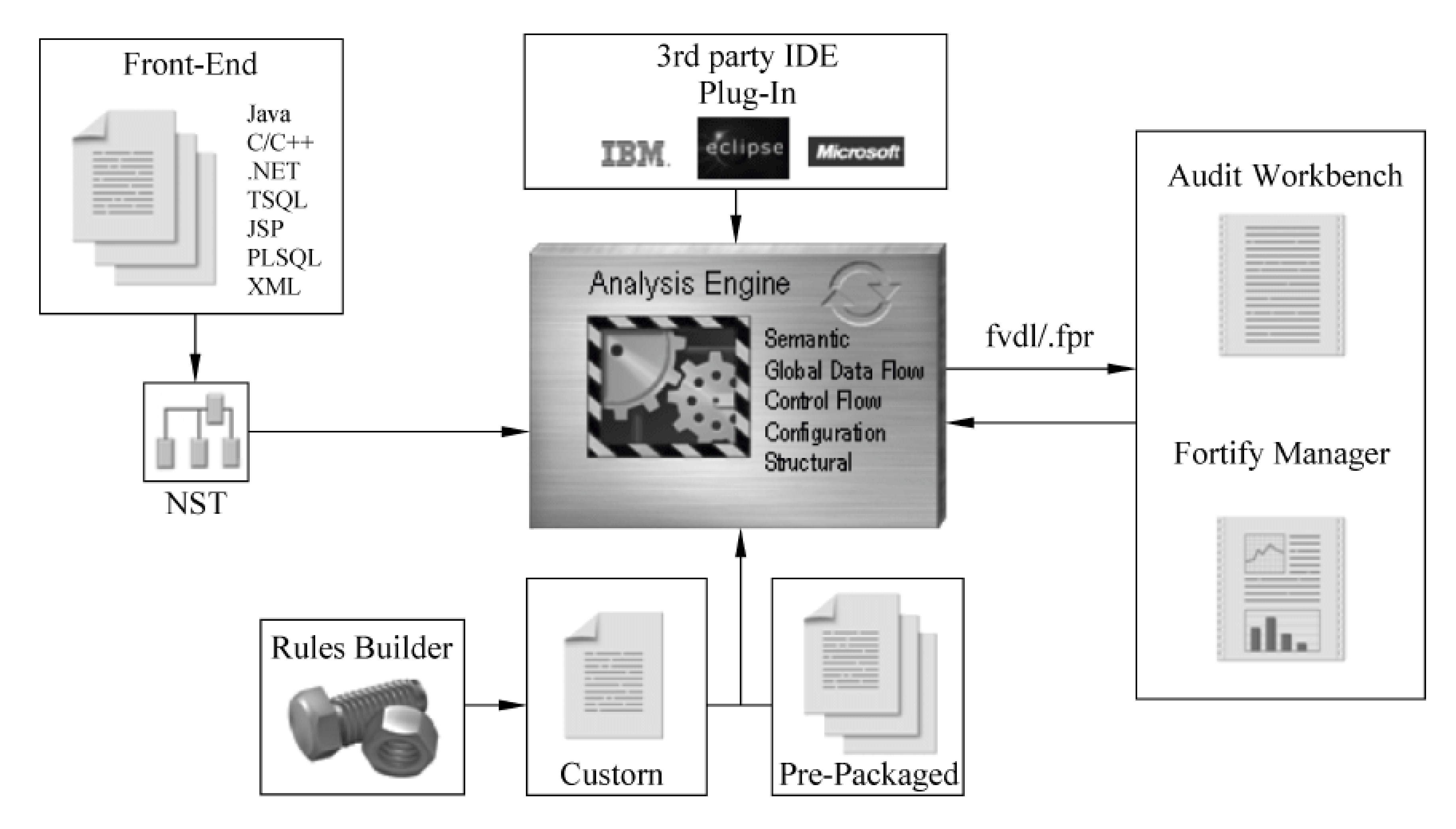


图 5-26　HP FortifySCA 工作原理

生的全路径、漏洞所在的源代码行、漏洞的详细说明及修复建议等,如图 5-27 所示。

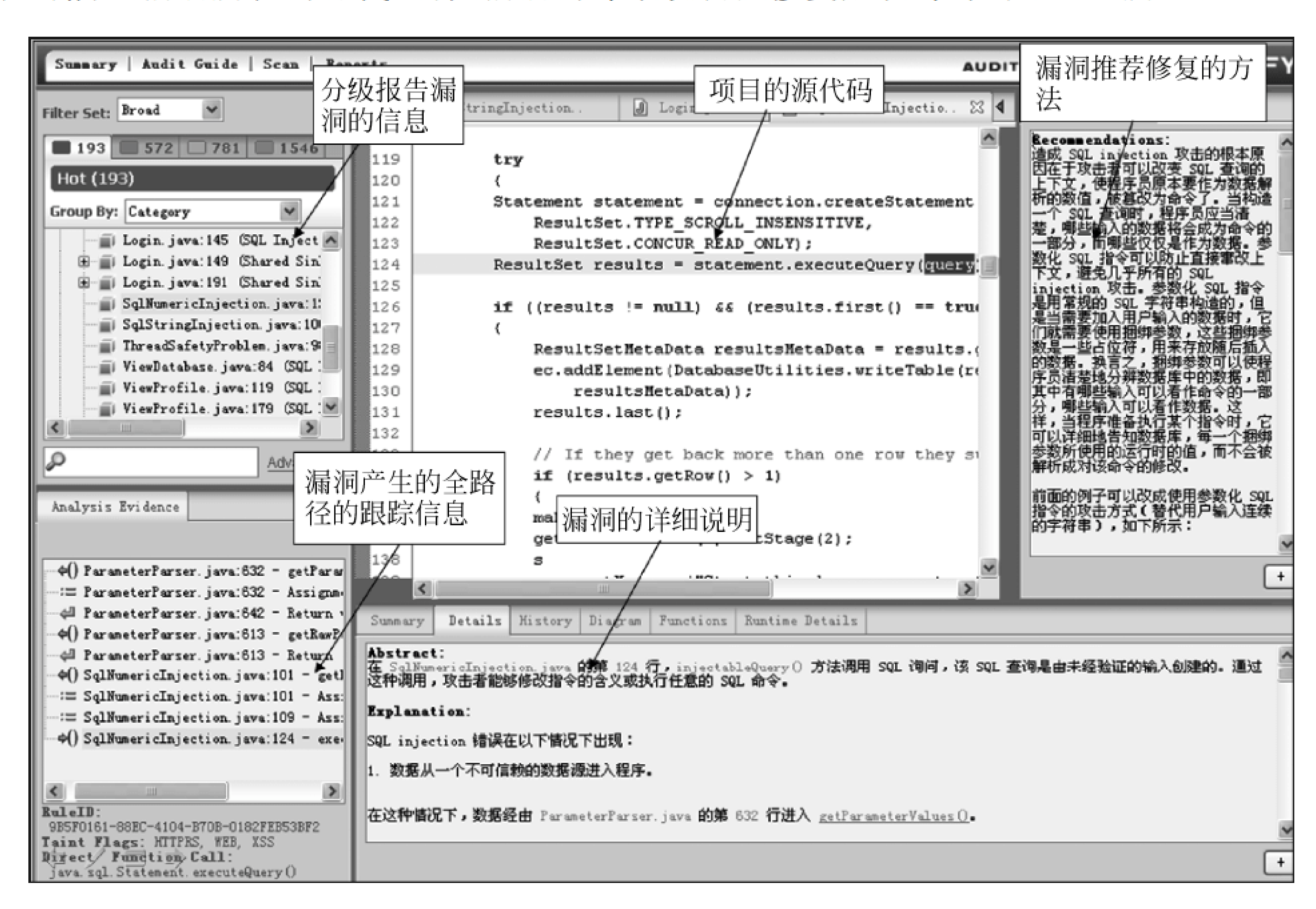


图 5-27　HP FortifyAWB 查看结果

5. HP FortifySCA 主要功能

HP FortifySCA 是一组软件安全分析器,能够在多种语言中搜索那些违背安全编码规则和指导原则的情况。由 HP FortifySCA 语言技术提供的丰富数据能够使这些分析器查明违背规则的情况并将其分级,从而可以快速准确地进行修复。这样不仅可以提供更加安全的软件产品,而且还有助于使安全代码的检查更加有效、一致而完整,特别是涉及大型代码库的情况。模块化的体系结构能使用户快速地上传新的、第三方的以及客户指定的安全规则。

概括来讲，使用 HP FortifySCA 会涉及这些操作：

① （可选操作）将 HP FortifySCA 集成到构建过程中；

② 针对代码库执行分为两个阶段的分析过程，最后产生一份安全漏洞报告；

③ （可选操作）将结果传送到 Audit Workbench 和 Fortify Team Server，进行分析与检查。

1）源代码分析功能

HP FortifySCA 包含五个不同的分析器：数据流分析器、控制流分析器、语义分析器、结构分析器以及配置分析器。每个分析器均会接受不同类型的规则，该规则经过特殊定制，可以为相应类型的分析提供必要的信息。规则是指用来识别源代码中可能引发安全漏洞或其他不安全因素的各种元素的定义。

规则根据所适用的分析器进行划分，于是就产生了针对数据流分析器、控制流分析器、语义分析器、结构分析器以及配置分析器的各种规则。规则类型还可以进一步划分，以反映问题的类别或者规则所代表的信息类型。

（1）数据流分析器可以检测涉及将被感染数据（用户控制的输入）用于危险用途的潜在漏洞。数据流分析器使用全局的、程序间的感染繁殖分析，检测 source（用户输入的站点）与 sink（危险的函数调用或者操作）之间的数据流。例如，数据流分析器可以检测一个用户控制的特别长的字符串输入是否正被复制到一个固定长度的缓冲区中，还可以检测一个用户控制的字符串是否正被用来构建 SQL 查询文本。

（2）控制流分析器可以检测潜在危险的操作执行顺序。通过分析程序中的控制流路径，控制流分析器能确定在执行一系列操作时是否遵循了特定的顺序。例如，控制流分析器可以检测 check/time 的时间和未经初始化的变量，并检查实用程序，如 XML 阅读器，是否在使用前做了正确的配置。

（3）语义分析器可以在程序内部这个层面检测可能会引发潜在危险的函数和 API 的各种使用情况。其特定的逻辑会搜索 buffer overflow、format string 和各种执行路径的问题，但并不局限于这几个类别。任何存在潜在危险的函数调用都可以通过语义分析器进行标记。例如，语义分析器可以检测 Java 中的过时函数和 C/C++ 中的不安全函数，如 gets()。

（4）配置分析器可以在应用程序的配置文件中搜索错误、缺陷和违反规则的策略漏洞。例如，配置分析器可检查网络应用程序的某一用户会话的超时是否合理。

（5）结构分析器可以检测可能存在危险的结构缺陷或程序定义。通过理解程序构建的方式，结构分析器能够识别出安全编程实践中违反规则的各种情况以及通常难以被检查、检测到的技术，原因是这些技术涉及的范围很广，包括有关声明和变量函数的使用。例如，结构分析器检测在 Java Servlets 中成员变量的赋值，识别未被声明为 static final 的记录器的使用，并标记那些由于断言条件始终不满足而永不被执行的 dead code 实例。

2）跨层、跨语言地分析整个项目

（1）跨项目的各个架构层。对于多层（三层、四层等）开发架构，能够跨越所有层来分析整个项目。

（2）跨项目使用的各个语言。对于包含多种开发语言的项目，能够所有的代码一起透明地分析。

3）支持多种语言、平台和开发环境

以 IDE 插件或者命令行执行的方式支持多种实施环境，全面地检测项目源代码中存在的安全漏洞。

(1) 支持的语言有 Java/JSP C/C++、. NET 平台、TSQL/PLSQL、Cold Fusion、XML、CFML、ASP、PHP、JS、VB。

(2) 支持的操作系统有 Windows、Solaris、Red Hat Linux、Mac OS X、HP-UX、IBM AIX。

4) 安全编码规则包

数十万条规则，覆盖业界最广泛约 400 多种安全漏洞，完整全面的源代码不安全编码特征，与 SCA 的分析引擎一同工作，查找源代码中的安全漏洞。

目前 HP FortifySCA 可以扫描出约 400 多种漏洞，HP Fortify 将所有安全漏洞整理分类，根据开发语言分项目，再细分为 8 个大类，约 400 多个子类，具体详细信息可登录 HP Fortify 官方网站(http://www. hpenterprisesecurity. com/vulncat/en/vulncat/index. html)。

5) 代码安全审查(Audit WorkBench)

(1) 说明漏洞原因、分级别报告漏洞、提供修复方法、报告漏洞产生的全路径(说明漏洞产生的原因，说明为什么会称之为漏洞及此类型漏洞有可能产生的危害。漏洞根据严重级别分为三种级别：Hot，Warning，Info。根据程序流程记录可以清楚漏洞产生的过程，并定位到对应的源代码，根据不同类别的漏洞给出有效的解决方法的说明文档)。

(2) 安全评审(对漏洞进行筛选、排序；对漏洞进行过滤，调整级别；对漏洞进行批注，标记评审状态)。

(3) 报表功能(提供多种报表模板，提供 PDF、Word、XML 的报表格式，可以选择报表中的具体内容，提供与 OWASP Top10 2007 的比较报表)。

6) 源代码分析过程

(1) 转换(通过一系列命令聚集起来的源代码会被转换为与一个构建 ID 相关联的中间格式。该构建 ID 通常就是正在扫描的项目的名称)；

(2) 验证可用的行数(一旦文件转换完毕，即验证可用的行数是否大于或等于对转换后的文件进行扫描所需的行数)；

(3) 分析(系统将扫描在转换阶段识别出的源文件，并生成一个分析结果文件，该文件通常为 HP Fortify 项目 (FPR) 格式。FPR 文件通过文件扩展名 fpr 来表示)；

(4) 对转换和分析阶段的验证(确保源文件的扫描使用了正确的规则包，而且没有报告重大的错误)。

习题

1. 什么是静态测试？静态测试包括哪些内容？
2. 什么是同行评审？简述同行评审的内容和流程。
3. 什么是需求规格说明测试？如何对需求规格说明进行评审？
4. 什么是代码审查？代码审查包括哪些内容？
5. 代码检查包括哪些内容？如何进行代码检查？
6. 什么是编码规范？确立和遵守编码规范有何意义？
7. 代码分析工具是怎样工作的？代码自动分析都有哪些内容？简单介绍几款针对不同语言的编程规则检查工具(在网上查找)。
8. 什么是代码结构分析？代码结构分析有何意义？
9. 从网上下载画程序控制流图和程序调用图的软件，并给出实际使用的例子和结果。
10. 什么是代码安全性检查？简述代码安全性检查的方法和内容。

11. 什么是软件复杂性？软件复杂性包括哪些内容？

12. 什么是 Halstead 复杂度？Halstead 复杂度度量的主要思想是什么？

13. McCabe 复杂度的中心思想是什么？如何进行 McCabe 复杂度的度量？

14. 简述软件复杂性度量的基本方法，采取何种手段控制软件复杂性？

15. 面向对象软件复杂性度量的特性有哪些？常用于度量的方法有哪些？

16. 简述软件质量定义，软件质量属性包括哪些内容？

17. 简述软件质量分层模型概念，目前流行的质量分层模型有哪些？

18. GB/T 16260—2006（软件工程 产品质量标准）质量模型中的质量特性/子特性有哪些，我们一般如何处理质量特性/子特性之间的冲突？

19. 如何度量和评价软件质量？简述高级语言的源代码质量度量和评价流程。

20. 简单介绍几款软件质量度量和评价的工具（在网上查找）。

21. 学习 Logiscope 软件质量评价模型，解释该质量模型的计算方法和具体应用方法。

22. 简述惠普静态分析工具的用途和功能。

第6章 软件动态测试

动态测试是指通过运行被测程序，检查运行结果与预期结果的差异，并分析运行效率和健壮性等。这种方法由三部分组成：构造测试用例、运行被测程序并执行测试用例、分析程序的输出结果。

对于动态测试，可以从不同的角度进行分类。如从是否关心软件内部结构和具体实现的角度划分，软件测试可以分为"白盒"测试、"黑盒"测试和"灰盒"测试；从软件开发的过程的角度，软件测试可以分为单元测试、集成测试、确认测试、系统测试、验收测试及回归测试；从测试执行时是否需要人工干预的角度划分，软件测试可以分为人工测试和自动化测试；从测试实施组织的角度划分，软件测试可分为开发方测试、用户测试(β测试)、第三方测试。

6.1 "白盒"测试

"白盒"测试是一种典型的测试方法，是一种按照程序内部逻辑结构和编码结构设计测试数据或测试用例并完成测试的测试方法，因此又称为结构测试或逻辑驱动测试。它是基于一个应用代码的内部逻辑知识，测试覆盖全部语句、分支、路径和条件。它利用查看代码功能和实现方式得到的信息来确定哪些需要测试、哪些不需要测试、如何展开测试。

"白盒"测试一般分为静态测试和动态测试，静态测试不实际运行软件，主要是对软件的编程格式、结构等方面进行评估，采用的是代码走查、代码审查、程序结构分析、控制流分析、数据流分析及信息流分析等；而动态测试需要在主机环境或目标机环境中实际运行软件，并使用设计的测试用例去探测软件缺陷。所采用的测试方法是逻辑覆盖(包括语句覆盖、分支覆盖、条件覆盖、分支-条件覆盖以及路径覆盖)。需要注意的是不要把动态"白盒"测试和调试弄混了。调试和动态"白盒"测试都包括处理软件缺陷和查看代码的过程，但是它们的目标不同，其中又有交叉，如图6-1所示。

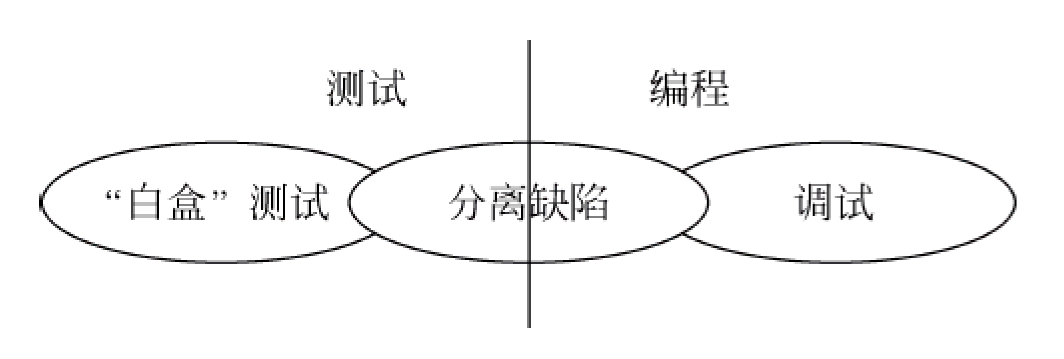


图6-1 调试和"白盒"测试目标的交叉

由图6-1可以看出动态"白盒"测试的目的是发现缺陷，而调试的目的是改正缺陷，但它们共同的目的是分离缺陷。

"白盒"测试特点主要有：①可以构成测试数据，使特定程序部分得到测试；②有一定的充分性度量手段；③可获得较多工具支持；④通常只用于单元测试。

"白盒"测试的内容有：①对程序模块的所有独立执行路径至少测试一次；②对所有的逻辑判定，取"真"与取"假"的两种情况都至少测试一次；③在循环的边界和边界内执行循环体；④测试内部数据结构的有效性。

我们用例6-1来讲述"白盒"测试的方法。

例 6-1

```
func(int a, b, x)
{
if ((a>1) && (b=0))
    x=x/a;
if ((a=2) || (x>1))
    x=x+1;
}
```

该程序的流程图如图 6-2 所示。

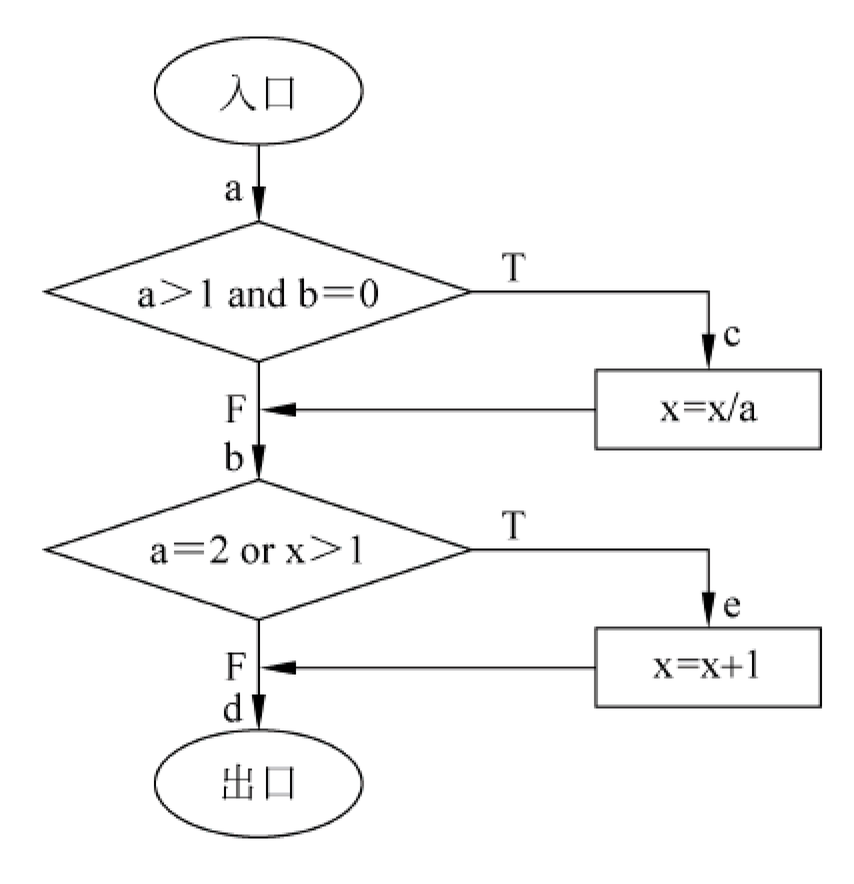


图 6-2 例子程序的流程图

6.1.1 逻辑覆盖

逻辑覆盖是以程序内部的逻辑结构为基础的测试方法,属“白盒”测试。这一方法是一系列测试过程的总称,它要求测试人员对程序的逻辑结构有清楚的了解。从覆盖源程序的各个方面考虑,大致可以分为语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。

1. 语句覆盖

为了暴露程序中的错误,语句覆盖是最起码的测试要求,要求设计足够多的测试用例,使得每一条语句至少被执行一次。对例 6-1 来说,只要选取 a=2,b=0,x=3,就可以达到每一条语句至少执行一次的要求。

语句覆盖对程序的逻辑覆盖很少,在上边的例子中,两个判定条件都只测试了条件为真的情况,如果条件为假时处理有误,显然不能发现。此外,语句覆盖只关心判定表达式的值,而没有分别测试判定表达式中每个条件取不同值时的情况。在上面的例子中,为了满足语句覆盖测试的要求,只需两个判定表达式(a>1)and (b=0) 和 (a=2)or (x>1) 都取真值,因此使用上述一组测试数据就够了。但是如果把程序中的第一个判定表达式中的逻辑运算符“and”写成“or”,或者把第二个判定表达式中的条件写成“x<1”,使用上边的测试数据则不能查出这些错误。

因此,语句覆盖是很弱的逻辑覆盖标准,为了更充分地测试程序,可以采用下边讲述的其他逻辑覆盖方法。

语句覆盖的优点:①检查所有语句;②结构简单的代码它的测试效果较好;③容易实现自动测试;④代码覆盖率高;⑤如果是程序块覆盖,则不用考虑程序块中的源代码。

语句覆盖不能检查出的错误有:

(1) 条件语句错误(如“a > 1 && b == 0” →“a > 0 && b == 0”);

(2) 逻辑运算(&&、||)错误(如“a > 1 && b == 0”→“a > 1 || b == 0”,“ u = a< 1 || b>2 ”→“ u = a<1 ”);

(3) 循环语句错误,如循环次数错误(for(i=0;i<10;i++) statement;→for(i=0;i<=10;i++) statement;)及跳出循环条件错误(while(x>3) statement; →(while(x>3&&x<7) statement;)。

2. 判定覆盖

判定覆盖又叫分支覆盖,要求设计足够多的测试用例,使得程序中的每一个分支至少通过一次,即每一条分支语句的“真”值和“假”值都至少执行一次。while 语句、switch 语句、异常处理、跳转语句和三目运算符(a? b: c)等同样可以使用分支覆盖来测试。对于例 6-1 的流程

图分支来说，设计两个测试用例即可使它能通过 acd 和 abe 路径就达到了分支覆盖。

例如：(1) a=3，b=0，x=1 (沿 acd 路径执行)

(2) a=2，b=1，x=3 (沿 abe 路径执行)

除了双分支语句外还有多分支语句，如 C 语言中的 case 语句，分支覆盖必须对每一个分支的每一种可能的结果都进行测试，但是上面的测试用例没有检查 abd 路径执行时，x 的值是否有变化。因此，判定覆盖虽然比语句覆盖强，但是对程序逻辑的覆盖程度仍然不高。

判定覆盖要比语句覆盖查错能力强一些：执行了分支覆盖，实际也就执行了语句覆盖。判定覆盖与语句覆盖存在同样的缺点：不能查出条件语句错误，不能查出逻辑运算错误，不能查出循环次数错误，不能查出循环条件错误。

3. 条件覆盖

条件覆盖是指选择足够的测试用例，使得运行这些测试用例后，要使每个判断中每个条件的可能取值至少满足一次，但未必能覆盖全部分支。

对于例 6-1 程序中(a>1)&&(b=0)包含两个条件：a>1，b=0；(a=2)||(x>1)包含两个条件：a=2，x>1。因此，流程图中共有四个条件：a>1，b=0，a =2，x>1。为了达到条件覆盖的要求，需要有足够的例子来满足在 a 点有 a>1，a<=1，b=0，b！=0 等各种结果，在 b 点有 a=2，a！=2，x>1，x<=1 等各种结果。因此可设计以下两个测试用例，来满足这一标准：

例如：(1)a=2，b=0，x=4(沿 ace 路径执行)

(2) a=1，b=1，x=1(沿 abd 路径执行)

同样是两个测试用例，但是这两个测试用例比分支覆盖中的更有效。因为它使判定表达式中每个条件都取到了两个不同的结果，判定覆盖只关心整个判定表达式的结果。例如，上边两组测试数据也同时满足了判定覆盖的要求。但是，也可能出现相反的情况：虽然每个条件都取到了不同的结果，判定表达式却有可能始终都是一个值。例如，如果使用下面两组测试用例，则只满足条件覆盖的要求而不满足判定覆盖的要求。

例如：(1) a=2，b=0，x=1(满足 a>1，b=0，a=2 和 x<=1 的条件，执行路径 ace)

(2) a=1，b=1，x=2(满足 a<=1，b！=0，a！=2 和 x>1 的条件，执行路径 abe)

条件覆盖的利弊：①能够检查所有的条件错误；②不能实现对每个分支的检查；③测试用例数增加。

4. 判定/条件覆盖

既然条件覆盖不一定包括判定覆盖，判定覆盖也不一定包括条件覆盖，自然会提出一种能够同时满足这两种覆盖标准的逻辑覆盖，这就是判定/条件覆盖。判定/条件覆盖就是设计足够多的测试用例，使得判定中每个条件的所有可能取值至少能够获取一次，同时每个判断的所有可能的判定结果至少执行一次。换言之，即是要求各个判定的所有可能的条件取值组合至少执行一次。

对于例 6-1 的程序而言，下述两组测试用例可以满足判定/条件覆盖的要求。

例如：

(1) a=2，b=0，x=4。

(2) a=1，b=1，x=1。

但是，这两组测试用例也就是为了满足条件覆盖标准最初选取的两组数据，因此，有时候判定/条件覆盖也并不比条件覆盖强。

分支-条件覆盖的利弊：①既考虑了每一个条件，又考虑了每一个分支，发现错误能力强于分支覆盖和条件覆盖；②并不能全面覆盖所有路径；③测试用例数量的增加。

5. 条件组合覆盖

要求设计足够多的测试用例，使得每个判定中条件的各种组合至少出现一次。

对于例 6-1 来说，程序中有四个条件：a>1，b=0，a=2，x>1。因此设计测试用例时，应满足下面八种条件组合：

① a>1，b=0；②a>1，b！=0；③a<=1，b=0；④a<=1，b！=0；⑤a=2，x>1；⑥a=2，x<=1；⑦a！=2，x>1；⑧a！=2，x<=1。

例如：

(1) a=1，b=0，x=2 满足③⑦组合。

(2) a=1，b=1，x=1 满足④⑧组合。

(3) a=2，b=0，x=4 满足①⑤组合。

(4) a=2，b=1，x=1 满足②⑥组合。

显然，满足条件组合覆盖标准的测试用例，也一定满足判定覆盖、条件覆盖和判定/条件覆盖标准。因此，条件组合覆盖是前面几种覆盖标准中最强的。但是，满足条件组合覆盖要求的测试用例并不一定能使程序中的每条路径都执行到，例如，上述 4 组测试用例都没有测试到路径 acbd。

以上根据测试数据对源程序语句检测的详尽程度，简单讨论了几种逻辑覆盖标准。在上边的分析过程中常常谈到测试数据执行的程序路径，显而易见，测试数据可以检测的程序路径的多少，也在一定程度上反映了对程序测试的详尽程度，也就是下边要讲的路径覆盖。

6. 路径覆盖

要求设计足够多的测试用例，使得程序中所有的路径都至少执行一次。针对例 6-1 的流程图来说，它有 4 条路径 ace，abd，abe，acd。因此设计了下面四个测试用例：

例如：

(1) a=2,b=0,x=3 覆盖 ace。

(2) a=2,b=1,x=1 覆盖 abe。

(3) a=1,b=0,x=1 覆盖 abd。

(4) a=3,b=0,x=1 覆盖 acd。

6.1.2 路径测试

路径测试就是根据程序的逻辑控制所产生的路径进行测试用例设计的方法。它是从一个程序的入口开始，执行所经历的各个语句的完整过程。从广义的角度讲，任何有关路径分析的测试都可以被称为路径测试。

完成路径测试的理想情况是做到路径覆盖，但对于复杂性大的程序要做到所有路径覆盖(测试所有可执行路径)是不可能的。

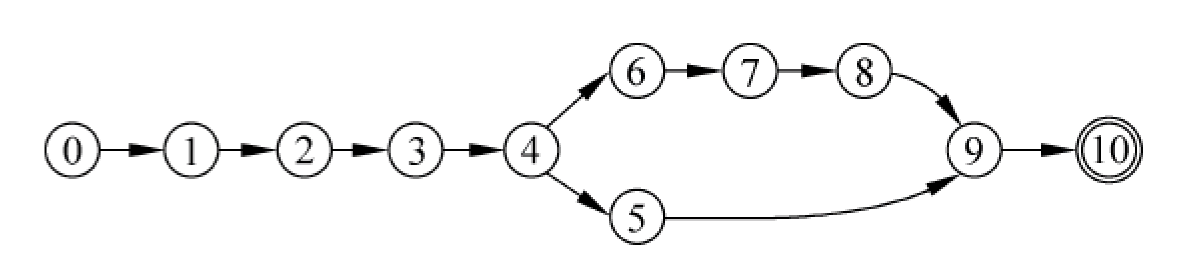
在不能做到所有路径覆盖的前提下，如果某一程序的每一个独立路径都被测试过，那么可以认为程序中的每个语句都已经检验过了，即达到了语句覆盖。这种测试方法就是通常所说的基本路径测试方法。

下面介绍几种常用的路径测试方法。

1. DD-路径测试

DD-路径(Decision-to-Decision Path)主要着眼命令式程序语言的测试覆盖率问题。程序有向图中存在分支，覆盖率考虑的是对各个分支情况的测试覆盖程度，因此对有向图中的线性串行部分进行压缩，在压缩图(即 DD-路径)的基础上进行测试用例设计，用测试覆盖指标考察测试效果。

例如下面图示的有向图：



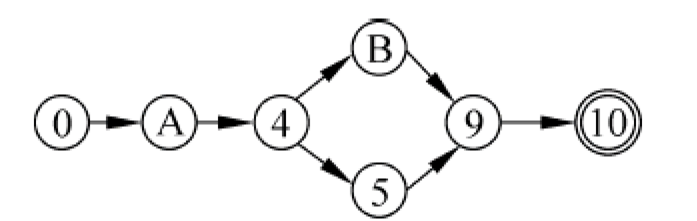


图 6-3　有向图到 DD-路径图的转化

对应的 DD-路径如图 6-3 所示。

即将节点 1,2,3 合并成节点 A,节点 6,7,8 合并成节点 B。合并原则为：将一系列邻接的顺序语句的节点合并。这样压缩的目的是将程序执行的分支情况清晰地提取出来,便于覆盖率的分析。

提出 DD-路径的目的：很多质量机构都把 DD-路径覆盖作为测试覆盖的最低可接受级别。E. F. Miller 发现,当通过一组测试用例满足 DD-路径覆盖要求时,可以发现全部缺陷中的大约 85%(Miller,1991)。

如果每一条 DD-路径都被遍历,则我们知道每个判断分支都被执行,其实就是遍历 DD-路径图中每条边。对于 if 类的语句,这意味着“真”、“假”分支都要覆盖。对于 CASE 语句,则每个子句都要覆盖。

2. 基本路径测试

例 6-1 是一个非常简单的程序段,只有 4 条路径。但是在实际问题中,一个不太复杂的程序,其路径都是一个非常庞大的数字。例如图 6-4 中所示的程序竟有 5^{20} 条路径。要想在测试中覆盖这许许多多的路径是不现实的。为了解决这一难题,只得把覆盖的路径数据压缩到一定限度内,例如,程序中的循环体只执行一次。这里所介绍的基本路径测试就是这样一种测试方法。

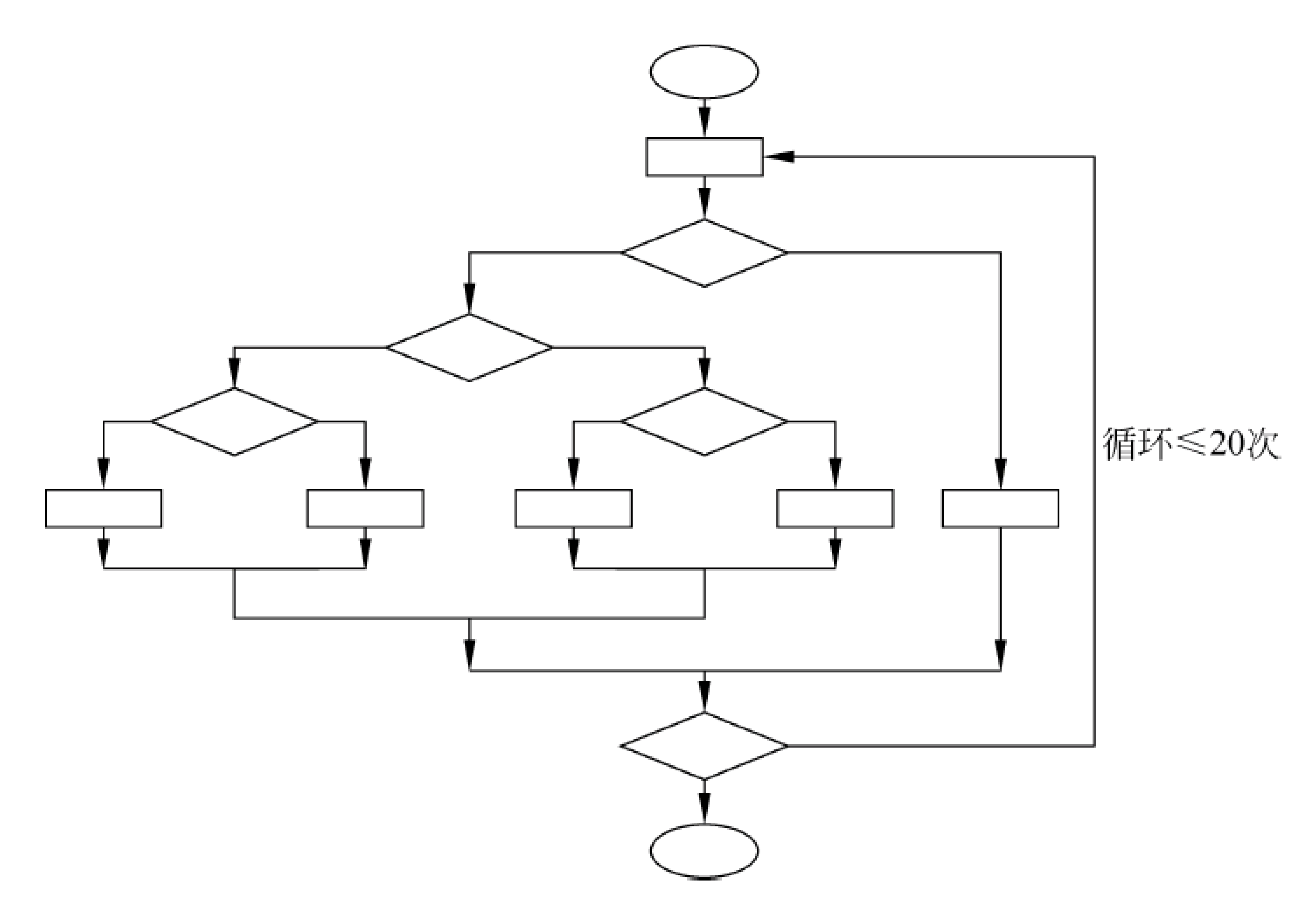


图 6-4　多次循环导致的天文路径数

基本路径测试是 McCabe 提出的一种“白盒”测试方法。使用这种方法设计测试用例时,首先要计算程序的圈复杂度,并以该复杂度为指南定义执行路径的基本集合,从该基本集合中导出的测试用例可以保证程序中的每条语句至少执行一次,而且每个条件在执行时都分别取“真”、“假”两种值。

使用基本路径测试方法设计测试用例的步骤如下。

(1) 根据过程设计结果画出相应的流图。

程序流图是描述程序控制流的一种图示方法。其中,基本的控制结构对应的图形符号如

图 6-5 所示。符号○称为控制流图的一个节点，它表示一个或多个无分支的源程序语句。

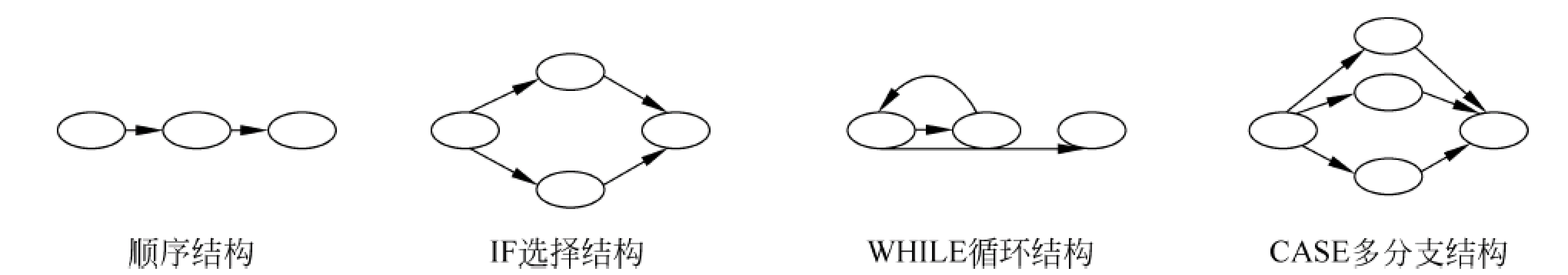


图 6-5 基本路径图的表示方法

例如，为了用基本路径测试方法测试下列用 PDL 语言描述的求平均值的过程，首先画出图 6-6 所示的流图。为了正确清楚地画出流图，我们把被映射为流图节点的语句编了序号。

利用 McCabe 圈复杂度的计算公式 V(G)＝m－n＋2P(V(G) 是强连通有向图 G 中的环数，m 是 G 中的弧数，n 是 G 中的节点数，p 是 G 中分离部分的数目，对于一个正常的程序来说，程序图总是连通的，即 p＝1，V(G)＝m－n＋2P，可进行基本路径计算。

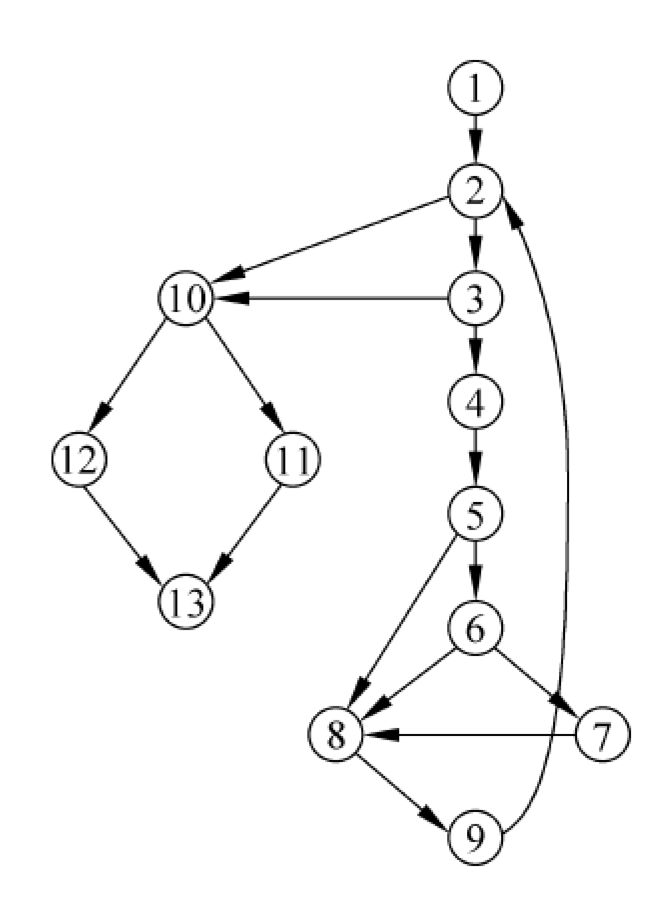


图 6-6 求平均值过程的流图

```
PROCEDURE average;
  /* 计算不超过 100 个在规定值域内的有效数字的平均值; 同
     时计算有效数字的总和及个数 */
  INTERFACE RETURNS average, total, input, valid;
  INTERFACE ACCEPTS value, minimum, maximum;

  TYPE value[1...100] IS SCALAR ARRAY;
  TYPE average, total, input, total.valid;
      minimum, maximum, sum IS SCALAR;
  TYPE i IS INTEGER;
1: i = 1;
   total.input = total.valid = 0;
   sum = 0;
2: DO WHILE value[i] <> -999
3:    AND total.input < 100
4: increment total.input by 1;
5: IF value[i] >= minimum
6:    AND value[i]<= maximum
7: THEN increment total.valid by 1;
        sum = sum + value[i];
8: ENDIF
   increment i by 1;
9: ENDDO
10:IF total.valid > 0
11:THEN average = sum/total.valid;
12:ELSE average = -999;
13:ENDIF
END average
```



(2) 计算控制流图的圈复杂度。

圈复杂度用于度量程序的逻辑复杂性。有了描绘程序控制流的流图之后，可以计算出流图的圈复杂度。经计算，图 6-6 所示流图的圈复杂度是 6。

(3) 确定线性独立路径的基本集合。

所谓的独立路径是指至少引入程序的一个新处理语句集合或一个新条件的路径,用流图术语描述,独立路径至少包含一条在定义该路径之前不曾用过的边。

使用基本路径测试方法设计测试用例时,程序的圈复杂度决定了程序中独立路径的数量,而且这个数是确保程序中所有语句至少被执行一次所需的最小测试数量。

对于图 6-6 所描述的求平均值的过程来说,由于圈复杂度为 6,因此有 6 条独立路径。例如,下面列出了几条独立路径。

路径 1:1—2—10—11—13

路径 2:1—2—10—12—13

路径 3:1—2—3—10—11—13

路径 4:1—2—3—4—5—6—8—9—2—…

路径 5:1—2—3—4—5—6—7—8—9—2—…

路径 6:1—2—3—4—5—8—9—2—…

路径后面的省略号表示,可以后接通过控制结构其余部分的任意路径。

通常在设计测试用例时,识别出判定节点是非常重要的。本例中的节点 2、3、5、6 和 10 就是判定节点。

(4) 设计可强制执行基本集合中每条路径的测试用例。

选取测试数据使得在测试每条路径时都适当地设置好各个判定节点的条件。例如,可以测试上一步得出的基本集合的测试用例如下。

路径 1 的测试用例:

value[k]=有效输入值,其中 k<i (i 的定义在下面)

value[i]= −999,其中 2 <= i <=100

预期结果:基于 k 的正确平均值和总数

其中,路经 1 无法独立测试,必须作为路径 4、5、6 的一部分来测试。

路径 2 的测试用例:

value[1]= −999

预期结果:average = −999,其他都保持初始值

路径 3 的测试用例:

试图处理 101 个或者更多个值

前 100 个数值应该是有效的输入值

预期结果:前 100 个数的平均值,总数为 100

其中,路径 3 同路径 1 一样,也是无法单独测试,必须作为路径 4、5、6 的一部分来进行测试。

路径 4 的测试用例:

value[i]=有效输入值,其中 i<100

value[k]>maximum,其中 k<i

预期结果:基于 k 的正确平均值和总数

在测试的过程中,执行每个测试用例并把实际输出结果与预期结果相比较。一旦执行完所有的测试用例,就可以确保程序中所有语句都被至少执行了一次,而且每个判定条件都分别取过 true 和 false 值。

3. 循环路径测试覆盖

循环路径测试分为:0 次循环(检查跳出循环),1 次循环(检查循环初始值),2 次循环(检

查多次循环),m 次循环(检查某次循环),最大次数循环,比最大次数多一次、少一次循环,检查循环次数边界。

循环使路径数量急剧增长,为此我们要对循环过程进行简化:无论循环的形式和实际执行循环体的次数多少,只考虑循环一次和 0 次。即进入循环体一次和跳出循环体。

路径覆盖的利弊:①实现了所有路径的测试,发现错误能力强;②某些条件错误可能无法发现;③路径数庞大,不可能覆盖所有路径;④用例数量增加。

6.1.3 数据流测试

数据流测试最初是因编译系统要生成有效的目标码而出现的,主要用于代码优化。现在主要用于发现定义/引用异常缺陷,如:变量被定义,但从来没有使用(引用);所使用的变量没有被定义;变量在使用之前被定义两次。

数据流测试与数据流图之间并没有什么关系,数据流测试关注变量赋值与使用位置,它是一种结构性测试方法,即可把它看做基于路径测试的一种改良方案(进行“真实性检查”)。数据流测试一方面仍会使用到路径测试的一些研究成果,另一方面也考虑向功能性测试目标靠近。

数据流测试重点关注的是变量的定义与使用测试。事实上,在调试修改 bug 时,我们也会经常这样做,例如在一段代码中搜索某个变量所有的定义、使用位置,思考在程序运行时该变量的值会如何变化,从而分析 bug 产生原因。数据流测试是将这种方法形式化,这样也便于构造算法,实现自动化分析。

数据流测试或称定义/使用测试是以程序数据流的视角(程序是一个程序元素对数据访问的过程),基于数据流关系(数据“定义-使用”对),使用程序图来描述数据“定义-使用”对,通过对路径进行“真实性检查”来发现数据不正确定义及使用问题,一种简单的数据流测试策略是要求覆盖每个定义-使用路径一次。

1. 定义-使用的相关定义

下面的定义中,P 代表程序,G(P)为程序图,V 为变量集合,P 的所有路径集合为 PATH(P)。

节点 n 是变量 v 的定义节点,记做 DEF(v,n)。如果执行对应这种语句的节点,那么与该变量关联的存储单元的内容就会改变。如输入语句、赋值语句、循环控制语句和过程调用,都是定义节点语句的例子。

节点 n 是变量 v 的使用节点,记做 USE(v,n)。如果执行对应这种语句的节点,那么与该变量关联的存储单元的内容会保持不变。如语句、赋值语句、条件语句、循环控制语句和过程调用,都是使用节点语句的例子。

例 6-2

```
a = b; DEF(1) = {a}, USES(1) = {b}.
a = a + b; DEF(1) = {a}, USES(1) = {a,b}.
```

如果 USE(v,n)是一个谓词使用(条件判断语句中),则记做 P-use;如果 USE(v,n)是一个运算使用(计算表达式中),则记做 C-use。

对应于谓词使用的节点永远有外度>=2,对应于计算使用的节点永远有外度<=1。

注:内度(即有向图中节点的内度)是将该节点作为终止节点的不同边的条数,外度(即有向图中的外度)是将该节点作为开始节点的不同边的条数。

变量 v 的定义-使用路径记做 du-path,如果 PATH(P)中的某个路径,如果定义节点 DEF(v,m)为该路径的起始节点,使用节点 USE(v,n)为该路径的终止节点,则该路径是 v 的定义-

使用路径。

变量 v 的定义-清除路径（Define-clear Path）记做 dc-path，如果变量 v 的某个定义-使用路径，除了起始节点之外没有其他定义节点，则该路径是变量 v 的定义-清除路径。

定义-使用路径和定义-清除路径描述了变量 v 从被定义的点到值被使用的点的源语句的数据流。通常不是定义-清除的定义-使用路径，有可能存在问题。

结合程序流图，找出所有变量的定义-使用路径，考察测试用例对这些路径的覆盖程度，就可以作为衡量测试效果的参考。例 6-3 给出了变量定义和使用分析的实例，如图 6-7 及表 6-1 所示。

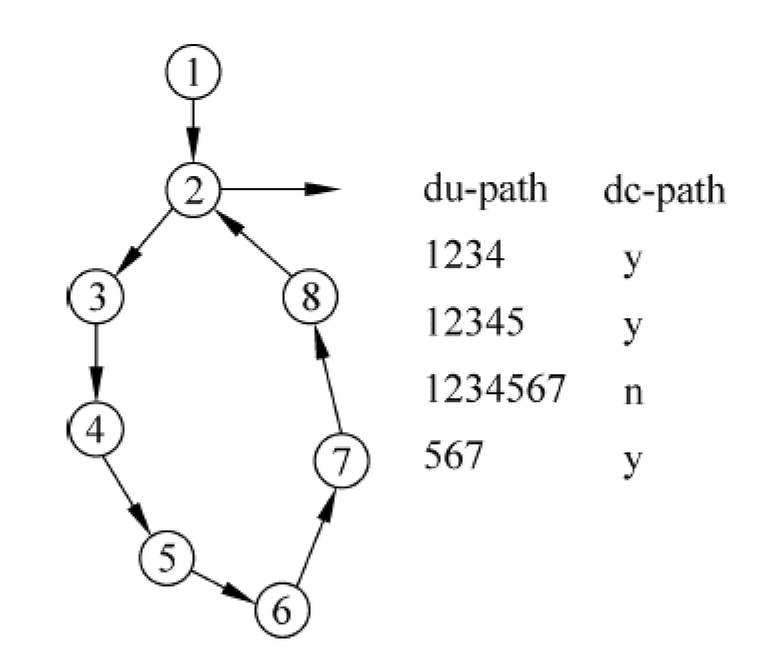


图 6-7 例 6-3 程序流图及定义-使用路径和定义-清除路径

表 6-1 例 6-3 中变量定义-使用及路径的情况列表

变量	定义节点	使用节点	路径（开始、结束）节点	是否定义清除
a	1，5	4，5，7	1，4　1，5　1，7　5，7	是　否　否　是
b	4	8	4，8	是

例 6-3

```
1 a = 5;                 // 定义 a
2 while(C1) {
3 if (C2){
4 b = a * a;             //定义 b,使用 a
5 a = a - 1;             //定义且使用 a
6 }
7 print(a);              //使用 a
8 print(b);}             //使用 b
```

2. 定义-使用的路径测试覆盖指标

对程序进行数据流分析的核心是定义一组叫做 Rapps-Weyuker 数据流覆盖指标，即根据 Rapps 和 Weyuker 所定义的一组基于数据流的测试路径覆盖指标，结合程序流图，找出所有变量的定义-使用路径，考察测试用例对这些路径的覆盖程度，以作为衡量测试效果的参考。

数据流覆盖指标假设所有程序变量都标识了定义节点和使用节点，且关于各变量都表示了定义-使用路径。并假定 T 为拥有变量集合 V 的程序 P 的程序图 G(P)中的一个路径集合。

（1）全定义（All-definition）覆盖准则：集合 T 满足程序 P 的全定义覆盖准则，当且仅当对于所有变量 v∈V，T 包含从 v 的每个定义节点到 v 的一个使用的定义清除路径。

（2）全使用（All-use）覆盖准则：集合 T 满足程序 P 的全使用覆盖准则，当且仅当对于所有变量 v∈V，T 包含从 v 的每个定义节点到 v 的所有使用的定义清除路径。

（3）全谓词使用（All Predicate use）/部分计算使用（Some Calculation Use）覆盖准则：集合 T 满足程序 P 的全谓词使用/部分计算使用准则，当且仅当所有变量 v∈V，T 包含从 v 的每个定义节点到 v 的所有谓词使用的定义清除路径，并且如果 v 的一个定义没有谓词使用，则到至少一个计算使用有一条定义清除路径。

（4）全计算使用（All C-use）/部分谓词使用（Some P-use）覆盖准则：集合 T 满足程序 P 的全计算使用/部分谓词使用准则，当且仅当所有变量 v∈V，T 包含从 v 的每个定义节点到 v 的

所有计算使用的定义清除路径,并且如果 v 的一个定义没有计算使用,则到至少一个谓词使用有一条定义清除路径。

(5) 全定义-使用路径(All Definition-use-paths, All-du-paths)覆盖准则:集合 T 满足程序 P 的全定义-使用路径准则,当且仅当所有变量 v∈V,T 包含从 v 的每个定义节点到 v 的所有使用的定义清除路径,并且这些路径要么有一次环经过,要么没有环路。

(6) 数据流覆盖指标 Rapps/Weyuker 层次结构如图 6-8 所示。

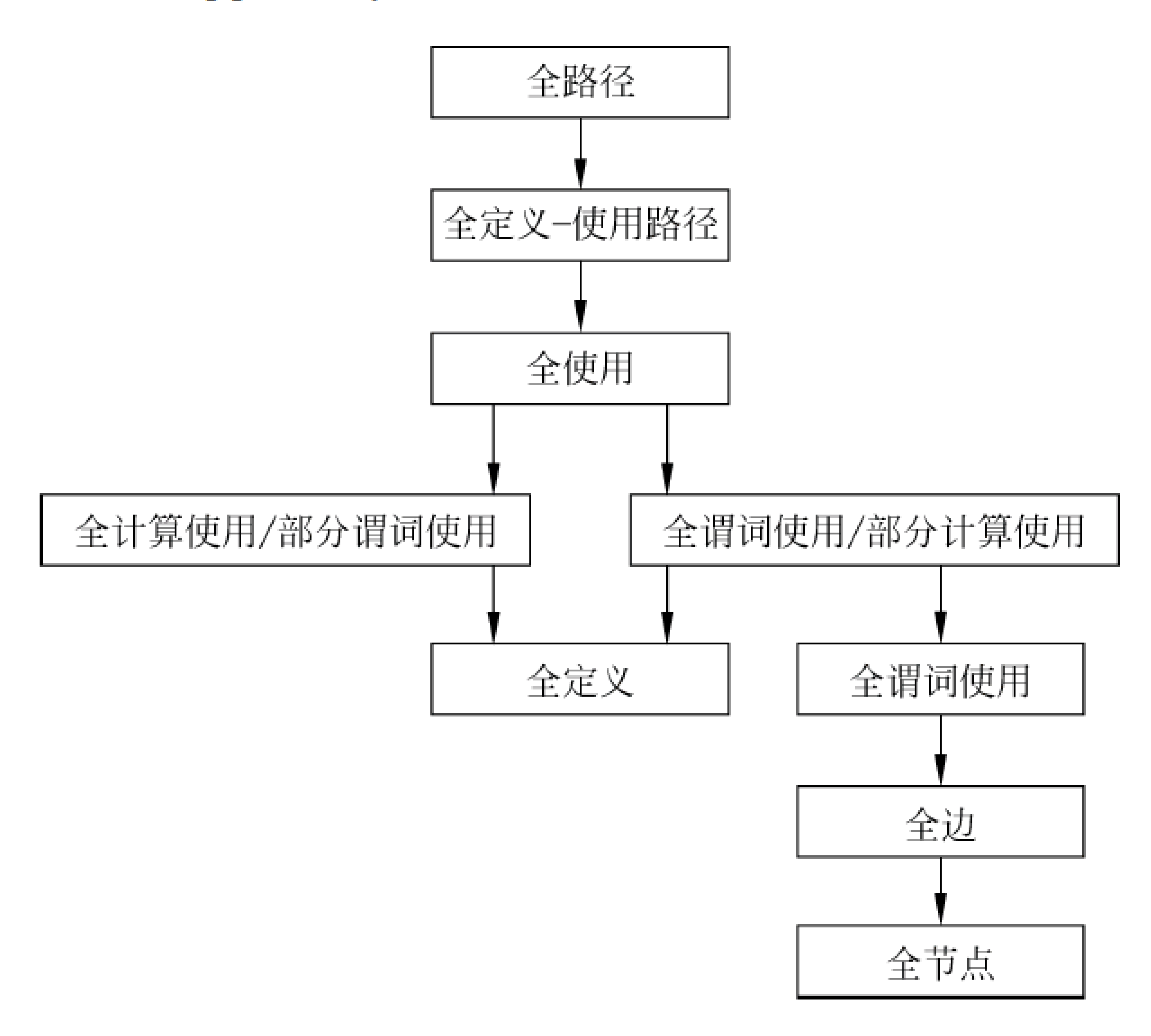


图 6-8 Rapps-Weyuker 数据流覆盖指标层次结构

在使用数据流覆盖指标时要注意全使用覆盖与全定义-使用覆盖的区别,如图 6-9 所示。

(1) 节点 b 定义 y,节点 c 和 d 使用 b 所定义的 y;

(2) 全使用覆盖虽然要求检查每个定义的所有可传递到的使用,但对如何从一个定义传递到一个使用不作要求;

(3) 全定义-使用覆盖要求检查所有可能的路径,但为了避免有环路时的无穷多条路径,限制只检查无环路的或只包含一条环路的路径。

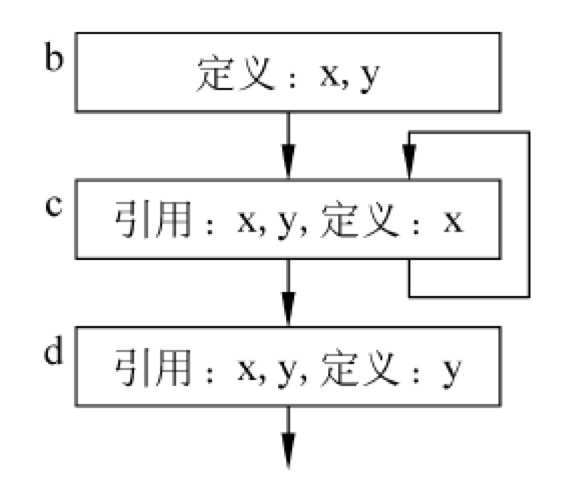


图 6-9 定义与使用示例

6.1.4 信息流分析

信息流分析是通过对输入数据、输出数据、语句之间关系(图 6-10)的分析来检查程序错误,还可用来分析是否存在无用的语句。下面我们通过例 6-4 来说明信息流分析的全过程。

例 6-4 整除算法例子

```
输入: in_m 是被除数
      in_n 是除数
输出: out_q 是商
      out_r 是余数
out_q = 0;
out_r = in_m;
while(out_r >= in_n)
{
```

```
4      out_q ++;
5      out_r = out_r - in_n;
    }
```

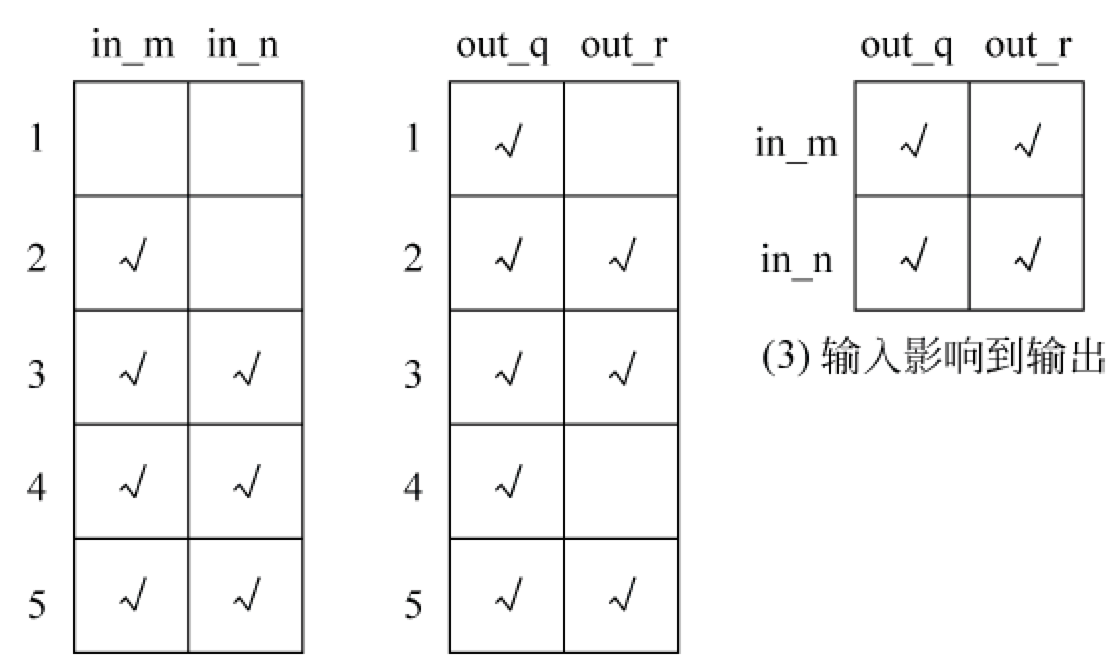


图 6-10 输入数据、输出数据、语句之间的关系

信息流分析的具体作用有三点：①能够列出对输入变量的所有可能的引用；②在程序的任何指定点检查其执行可能影响某一输出变量值的语句；③为输入、输出关系提供一种检查，看每个输出值是否有相关的输入值，而不是其他值导出。

6.1.5 覆盖率分析及测试覆盖准则

代码覆盖率是指采用“白盒”法进行测试时，考虑测试用例对程序内部逻辑的覆盖程度。最彻底的“白盒”法是覆盖程序中的每一条语句、每一个分支及每一条路径，但这往往无法实现。因此，需要采用一些标准来度量覆盖的程度，并希望覆盖程度尽可能高些。

覆盖率分析主要对代码的执行路径覆盖范围进行评估，这些覆盖是从不同要求出发，为设计测试用例提出依据的。软件人员进行覆盖测试，主要想对程序模块进行这些方面的检查：①对程序模块所有独立的执行路径至少测试一次；②对所有的逻辑判定，取“真”与取“假”这两种情况都至少测试一次；③在循环的边界和循环界内执行循环体；④测试内部数据结构的有效性。

覆盖测试实施过程中，我们要注意如下几个问题：

(1) 内部动作是否按照规格说明书的规定正常进行(按照程序内部的结构测试程序，检验程序中的每条通路是否都能按预定要求正确工作，而不顾它的功能)；

(2) 覆盖测试的主要方法(主要方法有逻辑驱动、基本路径测试等，主要用于软件验证)；

(3) 覆盖法要求全面了解程序内部逻辑结构，对所有逻辑路径进行测试(覆盖法是穷举路径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据)；

(4) 不同软件的覆盖率分析对应的覆盖测试方法是可以有差别的(覆盖率方法有很多，取决于对软件的要求程度，比如航空、医疗软件要求严格，需使用 DO-178B 的修正条件/判定覆盖或称 MC/DC 覆盖率标准)；

(5) 覆盖测试只是根据程序的内部结构进行测试，而不考虑外部特性(如果程序结构本身有问题，比如说程序逻辑有错误或是有遗漏，那是无法发现的)。

1. 逻辑覆盖率计算方法

逻辑覆盖率主要是指语句覆盖率、判定覆盖率、条件覆盖率、判定/条件覆盖率、条件组合

覆盖率和路径覆盖率。其计算方法是：

覆盖率＝(至少被执行一次的 item 数)/item 的总数

这个公式是对 item 的覆盖情况进行计算，item 可以是需求、语句、分支、条件、路径等。覆盖率是用来度量测试完整性的一个手段，不是测试的目的。通过覆盖率数据，可以知道测试是否充分，测试的弱点在哪些方面，进而指导我们去设计能够增加覆盖率的测试用例。

1) 语句覆盖率

语句覆盖率是指已执行的可执行语句占程序中可执行语句总数的百分比。计算公式如下：

语句覆盖率 ＝(至少被执行一次的语句数量)/(可执行的语句总数)

注意，对 C 语言，可执行的语句不包括：以＃开头的＃include、宏定义、预处理语句和注释语句。

```
if(x!= 1)
{                                      测试用例
     statements;
     …}99 句                           x = 2
}                                      语句覆盖率 99 %
else                                   50 % 的分支没有达到
{
     statement;
}
```

复杂的程序不可能达到语句的完全覆盖，当然语句覆盖率越高越好。

通常语句覆盖率可能出现这样的问题：语句覆盖率可能会很高，却有严重缺陷(如上面示例所示)。

2) 分支覆盖率

分支覆盖率是指已取过“真”和“假”两个值的判定占程序中所有条件判定个数的百分比。计算公式如下：

判定覆盖率＝(判定结果至少被评价一次的数量)/(判定结果的总数)

3) 条件覆盖率

条件覆盖率是指在测试时运行被测试程序后，所有判断语句中每个条件的可能取值(真值和假值)出现过的比率。计算公式如下：

条件覆盖率＝(条件操作数值至少被评价一次的数量)/(条件操作数值的总数)

条件操作数是条件的具体取值(“真”或者“假”)。

4) 分支-条件覆盖率或判断-条件覆盖率

分支-条件覆盖率是指在运行被测试程序后，所有判断语句中每个条件的所有可能值(为“真”为“假”)和每个判断本身的判定结果(为“真”为“假”)出现的比率。计算公式如下：

分支-条件覆盖率＝(条件操作数值或判定结果至少被评价一次的数量)/(条件操作数值总数＋判定结果总数)

5) 路径覆盖率

路径覆盖率是指在测试时，运行被测试程序后程序中所有可能的路径被执行过的比率。计算公式如下：

路径覆盖率＝(至少被执行到一次的路径数)/(总的路径数)

注意：N 个测试用例最多执行 N 条路径。

分支覆盖可以保证语句覆盖，条件覆盖和分支覆盖不能互相保证，路径覆盖可以保证分支

覆盖,路径覆盖不能保证条件覆盖。通常要求用路径覆盖做到语句覆盖和分支覆盖;在考虑程序中错误(异常)处理工作的重要性以及其结构相对简单时,要求错误处理要做到路径覆盖;对质量要求高的软件,可根据情况提出条件覆盖、分支-条件覆盖以及路径覆盖要求。

2. 覆盖准则

测试覆盖准则目前常用到的主要有 ESTCA 和 LJSAJ。

1) FOSTER 的 ESTCA 覆盖准则

事实上覆盖率达到 100%是很难的,我们经常要借助我们的经验针对性地对容易发生问题的地方设计测试用例。

K. A. Foster 从测试工作实践的教训出发,吸收了计算机硬件的测试原理,提出了一种经验型的测试覆盖准则,较好地解决了上述问题。

Foster 的经验型覆盖准则是从硬件的早期测试方法中得到启发的。我们知道,硬件测试中,对每一个门电路的输入、输出测试都是有额定标准的。通常,电路中一个门的错误常常是"输出总是 0",或是"输出总是 1"。与硬件测试中的这一情况类似,我们常常要重视程序中谓词的取值,但实际上它可能比硬件测试更加复杂。Foster 通过大量的实验确定了程序中谓词最容易出错的部分,得出了一套错误敏感测试用例分析(Error Sensitive Test Cases Analysis, ESTCA)规则。事实上,规则十分简单。

规则 1:对于 A rel B(rel 可以是 <、= 和 >)型的分支谓词,应适当地选择 A 与 B 的值,使得测试执行到该分支语句时,A < B, A = B 和 A > B 的情况分别出现一次。

规则 2:对于 A rel1 C(rel1 可以是 > 或是 <,A 是变量,C 是常量)型的分支谓词,当 rel1 为 < 时,应适当地选择 A 的值,使 A = C−M(M 是距 C 最小的正数,若 A 和 C 均为整型时,M = 1)。同样,当 rel1 为>时,应适当地选择 A,使 A = C+M。

规则 3:对外部输入变量赋值,使其在每一测试用例中均有不同的值与符号,并与同一组测试用例中其他变量的值与符号不一致。

显然,上述规则 1 是为了检测 rel 的错误,规则 2 是为了检测"差一"之类的错误(如本应是"IF A > 1"而错成"IF A > 0"),而规则 3 则是为了检测程序语句中的错误(如把引用一变量错写成引用一常量)。

上述三规则并不是完备的,但在普通程序的测试中确是有效的。原因在于规则本身是针对程序编写人员容易发生的错误,或是围绕着发生错误的频繁区域,从而提高了发现错误的命中率。

2) Woodward 等人的层次 LCSAJ 覆盖准则

Woodward 等人曾经指出结构覆盖的一些准则,如分支覆盖或路径覆盖,都不足以保证测试数据的有效性。为此,他们提出了一种层次 LCSAJ 覆盖准则。

LCSAJ 覆盖(Linear Code Sequence and Jump Coverage)即是线性顺序代码和跳转代码覆盖,是 Woodward 等人提出来的一套覆盖率准则。一个 LCSAJ 是一组顺序执行的代码,以控制流跳转为其结束点。它的定义如下:

(1) 它起始于程序的入口或者是一个可能导致控制流跳转的点。

(2) 它结束于程序的出口或者是一个可能导致控制流跳转的点。

(3) 对于该点,一个跳转在后面的序列中产生。

它不同于 DD-Path。DD-Path 是根据程序有向图决定的。一个 DD-Path 是指两个判断之间的路径,但其中不再有判断。程序的入口、出口和分支节点都可以是判断点。而 LCSAJ 的起点是根据程序本身决定的。它的起点是程序第一行或转移语句的入口点,或是控制流可

以跳转到达的点。因此,几个 LCSAJ 首尾相接构成 LCSAL 串,组成程序的一条路径。第一个 LCSAJ 起点为程序起点,最后一个 LCSAJ 终点为程序终点。一条程序路径可能是由两个、三个或多个 LCSAJ 组成的。基于 LCSAJ 与路径的这一关系,Woodward 提出了 LCSAJ 覆盖准则,这是一个分层的覆盖准则。

第一层:语句覆盖。

第二层:分支覆盖。

第三层:LCSAJ 覆盖。即程序中的每一个 LCSAJ 都至少在测试中经历一次。

第四层:两两 LCSAJ 覆盖。即程序中每两个首尾相连的 LCSAJ 组合起来在测试中都要经历一次。

第 n+2 层:每 n 个首尾相连的 LCSAJ 组合在测试中都要经历一次。

这说明了,越是高层的覆盖准则越难满足。在实施测试时,若要实现上述的 Woodward 层次 LCSAJ 覆盖,需要产生被测程序的所有 LCSAJ。

尽管 LCSAJ 覆盖比判定覆盖复杂得多,但是 LCSAJ 覆盖的自动化实现相对还是容易获得的。另外,对一个模块的微小的变动可能对 LCSAJ 产生重大影响,因此维护 LCSAJ 的测试数据是相当困难的。一个大模块包含极其庞大的 LCSAJ,因此要获得 100%的覆盖率也是不现实的。然而 Woodward 等人提供的证据表明,把测试 100%的 LCSAJ 作为目标比 100%的判定覆盖要有效得多。

6.2 "黑盒"测试

"黑盒"测试,又称为功能测试或数据驱动测试,是把测试对象当做看不见内部的黑盒。在完全不考虑程序内部结构和处理过程的情况下,测试者仅依据程序功能的需求规范考虑确定测试用例和推断测试结果的正确性。软件工程师使用"黑盒"测试可以导出执行程序所有功能需求的输入条件集,实现功能覆盖。功能覆盖中最常见的需求覆盖,通过设计一定的测试用例,要求每个需求点都被测试到。因此,根据软件产品需求规格说明中的功能设计规格,在计算机上进行测试,以证实每个实现了的功能是否符合要求。

"黑盒"测试意味着要在软件的接口处进行测试,它着眼于程序外部结构,不考虑内部逻辑结构,主要针对软件界面和软件功能进行测试。因此,可以说"黑盒"测试是站在使用软件或程序的角度,从输入数据与输出数据的对应关系出发进行的测试(如图 6-11 所示)。

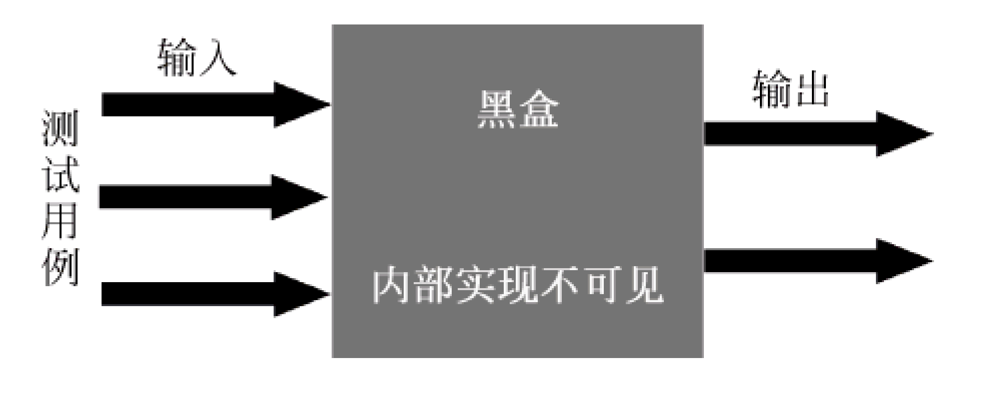


图 6-11 "黑盒"测试模型

很明显,如果外部特性本身设计有问题或规格说明的规定有误,用"黑盒"测试方法是发现不了的。因此,"黑盒"测试不能替代"白盒"测试,而是用来发现"白盒"测试以外的其他类型的错误,比如:①功能不对或遗漏;②接口错误或界面错误;③数据结构或外部数据库访问错误;④性能错误;⑤初始化和中止错误。

"黑盒"测试直观的想法就是既然程序被规定做某些事,那我们就看看它是不是在任何情况下都做得对。即用"黑盒"测试发现程序中的错误,须在所有可能的输入条件和输出条件下确定测试数据,检查程序是否都能产生正确的输出。很显然这是不可能的,因为穷举测试数量太大,人们不仅要测试所有合法的输入,而且还要对那些不合法但是可能的输入进行测试,无法完成。

假设一个程序 P 有输入量 X 和 Y 及输出量 Z。在字长为 32 位的计算机上运行。若 X、Y 取整数,按"黑盒"方法进行穷举测试,可能采用的测试数据组:$2^{32} \times 2^{32} = 2^{64}$。

如果测试一组数据需要 1 毫秒,一年工作 365×24 小时,完成所有测试需 5 亿年。

为此"黑盒"测试也要有一套产生测试用例的方法,用以产生有限的测试用例而覆盖足够多的"任何情况"。由于"黑盒"测试不需要了解程序内部结构,所以许多高层的测试如确认测试、系统测试、验收测试都采用"黑盒"测试。

用黑盒测试发现程序中的错误,必须在各种可能的输入条件和输出条件中确定测试数据,以检查程序是否都能产生正确的输出。黑盒测试所考虑的不是控制结构而是关注于信息域。它主要回答这几方面的问题:①如何测试功能的有效性?②如何测试系统行为和性能?③何种类型的输入会产生好的测试用例?④系统是否对特定的输入值特别敏感?⑤如何分隔数据类的边界?⑥系统能够承受何种数据率和数据量?

"黑盒"测试首先要求每个软件特性或功能必须被一个测试用例或一个被认可的异常所覆盖。另外,要求构造数据类型和数据值的最小集测试,即用一系列真实的数据类型和数据值运行,测试超负荷、饱和及其他"最坏情况"的结果;用假想的数据类型和数据值运行,测试排斥不规则输入的能力。最后,对影响性能的关键模块(如基本算法),应测试模块性能(包括精度、时间、容量等)。此时不仅要考核"程序是否做了该做的",还要考察"程序是否没做不该做的";同时还要考察程序在其他一些情况下是否正常。这些情况包括数据类型和数据值的异常等。

应用"黑盒"测试方法,所设计出的测试用例集都要满足以下两个标准:

(1) 所设计出的测试用例能够减少为达到合理测试所需要设计的测试用例的总数。

(2) 所设计出的测试用例能够告诉我们,是否存在某些类型的错误,而不是仅仅指出与特定测试有关的错误是否存在。

"黑盒"测试只有测试通过和测试失败两种结果,在进行通过测试时,实际上是确认软件能做什么,而不会去考验其能力如何。测试人员只运用最简单、最直观的测试用例。在设计和执行测试用例时,总是先要进行通过测试,即在进行破坏性试验之前,看一看软件基本功能是否能够实现。在确信了软件正确运行之后,可采取各种手段通过搞"垮"软件来找出缺陷。

"黑盒"测试与软件如何实现无关,如果实现发生变化,"黑盒"测试用例仍然可用(可重用性,面向回归测试)。另外,"黑盒"测试用例开发可以与软件开发同时进行,这样可节省软件开发时间,通过软件的用例(Use Case)就可以设计出大部分"黑盒"测试用例。

当然,"黑盒"测试也存在一些问题:测试用例数量较大,测试用例可能产生很多冗余,而且功能性测试的覆盖范围不可能达到 100%。

"黑盒"测试主要用到的方法有等价类划分法、因果图方法、边界值分析法、猜错法、随机数法等,这些测试方法是从更广泛的角度来进行"黑盒"测试。每种方法都力图能涵盖更多的"任何情况",但又各有长处。综合使用这些方法,会得到一个较好的测试用例集。

6.2.1 等价类划分

等价类划分法是典型的"黑盒"测试方法,该方法设计测试用例时完全不必考虑软件结构,只需考虑需求规格说明书中的功能要求。等价类划分法是把程序的输入域划分成若干部分,然后从每个部分中选取少数有代表性数据当做测试用例。每一类的代表性数据在测试中的作用等价于这一类中的其他值,也就是说,如果某一类中的一个例子发现了错误,这一等价类中的其他例子也能发现同样的错误;反之,如果某一类中的一个例子没有发现错误,则这一类中的其他例子也不会查出错误。使用这一方法设计测试用例,首先必须在分析需求规格说明的

基础上划分等价类，列出等价类表。

使用等价类划分法设计测试方案首先要划分输入数据的等价类，为此需要研究程序的功能说明。等价类实际上就是某个输入域的一个子集合，在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。等价类的划分有两种不同的情况：有效等价类和无效等价类。有效等价类是指对于程序的规格说明来说，是合理的、有意义的输入数据构成的集合；无效等价类是指对于程序的规格说明来说，是不合理的、无意义的输入数据构成的集合。在设计测试用例时，要同时考虑有效等价类和无效等价类的设计。

在确定输入数据的等价类时还常常需要分析输出数据的等价类，以便根据输出数据的等价类导出对应的输入数据等价类。

划分等价类需要经验，下面结合具体事例给出几条确定等价类的原则。

(1) 如果规定了输入条件的范围，则可以划分出一个有效等价类和两个无效等价类。例如，在程序的规格说明中，对输入条件有一个规定：

"…输入数值的范围是 1—999 …"

则有效等价类是"1＜＝ 输入数值 ＜＝ 999"，两个无效等价类是 "输入数值 ＜ 1"和"输入数值 ＞ 999"。

(2) 如果输入条件规定了输入值的集合，或者是规定了"必须如何"的条件，这时可以确定一个有效等价类和一个无效等价类。例如，在某一种语言中规定了变量标识符是以字母打头的字符串，那么所有以字母打头的标识符构成了有效等价类，而不在此集合内的构成了无效等价类。

(3) 如果输入条件是布尔值，那么可以确定一个有效等价类和一个无效等价类。

(4) 如果规定了输入数据的一组值，而且程序对不同输入值做不同的处理，则每个允许的输入值是一个有效等价类，此外还有一个无效等价类(任何一个不允许的输入值)。例如，在教师分房方案中规定对教授、副教授、讲师和助教分别计算分数，做相应的处理。因此可以确定 4 个有效等价类，分别为教授、副教授、讲师和助教，以及一个无效等价类，是所有不符合以上身份的人员输入值的集合。

(5) 如果规定了输入数据必须遵守的规则，则可以确立一个有效等价类(符合规则)和若干个无效等价类。例如，在 C 语言中，处理时规定"一个语句必须以分号'；'结束"。这时，就可以确定一个有效等价类"以'；'结束"，若干个无效等价类"以'，'结束"、"以'。'结束"、"以'：'结束"等。

(6) 如果确定知道已划分的等价类中各元素在程序中的处理方式不同，则应将此等价类进一步划分成更小的等价类。

以上这些规则只是测试时可能遇到的情况中的很小的一部分，实际的情况是千变万化的，根本不可能一一列出。为了能够正确地划分等价类，一是要注意经验积累，二是要正确分析被测程序的功能。此外，在划分无效等价类时，还要考虑编译程序的检错功能，最后再说明一点，上边这些规则虽然是针对输入数据的，但对于输出数据也同样适用。

在确立了等价类之后，建立等价类表，列出所有划分出的等价类，见图 6-12。从划分出的等价类中按以下原则选择测试用例：

(1) 为每一个等价类规定一个唯一编号；

(2) 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止；

(3) 设计一个新的测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直

到所有的无效等价类都被覆盖为止。

输入条件	有效等价类	无效
…	…	…
…	…	…

图 6-12 等价类表

下面是用等价类划分法设计一个简单程序的测试方案。

假设有一个把数字串转变为整数的函数。运行程序的计算机字长 16 位，用二进制补码表示整数。这个函数用 C 语言编写的，它的说明如下：

```
int strtoint(char shortstr[ ])
```

其中，参数 shortstr[]的定义如下：

```
char shortstr[6];
```

被处理的数字串是右对齐的，也就是说，如果数字串比 6 个字符短，则在它的左边补空格。如果字符串是负的，则负号和最高位数字紧相邻。因为编译程序固有的检错功能，测试时不需要使用长度不等于 6 的数组做实在参数，更不需要使用任何非字符数组类型的实在参数。通过分析这个程序的规格说明，可以划分出如下等价类。

有效输入的等价类：

(1) 1～6 个数字字符组成的数字串(最高位数字不是零)。

(2) 最高位数字是零的数字串。

(3) 最高位数字左邻是负号的数字串。

无效输入的等价类：

(4) 空字符串(全是空格)。

(5) 左部填充的字符既不是零也不是空格。

(6) 最高位数字右面由数字和空格混合组成。

(7) 最高位数字右面由数字和其他字符混合组成。

(8) 负号与最高位数字之间有空格。

合法输出的等价类：

(9) 在计算机能表示的最小负整数和零之间的负整数。

(10) 零。

(11) 在零和计算机能表示的最大正整数之间的正整数。

非法输出的等价类：

(12) 比计算机能表示的最小负整数还小的负整数。

(13) 比计算机能表示的最大正整数还大的正整数。

根据上面划分出的等价类，可以设计出如下测试用例：

(1) 1～6 个数字组成的数字串，输出是合法的正整数。

输入：‘ 1’；预期的输出 1。

(2) 最高位数字是零的数字串，输出是合法的正整数。

输入：‘000001’；预期的输出 1。

(3) 负号与最高位数字紧相邻，输出合法的负整数。

输入：‘－00001’；预期的输出－1。

(4) 高位数字是零,输出是零。

输入:'000000';预期的输出 0。

(5) 太小的负整数。

输入:'-47561';预期的输出:"错误——无效输入"。

(6) 太大的正整数。

输入:"134567";预期的输出:"错误——无效输入"。

(7) 空字符串。

输入:' ';预期的输出:"错误——没有数字"。

(8) 字符串左部字符既不是零也不是空格。

输入:'#####1';预期的输出:"错误——填充错误"。

(9) 最高位数字后有空格。

输入:' 1 2';预期的输出:"错误——无效的填充"。

(10) 最高位数字后面有其他字符。

输入:' 1##2';预期的输出:"错误——无效输入"。

(11) 负号和最高位数字之间有空格。

输入:' - 23';预期的输出:"错误——负号位置错误"。

表 6-2 所示为数字串转变为整数的等价类表。

表 6-2 数字串转变为整数的等价类表

输入条件	有效等价类	无效等价类
数字串个数	1 到 6 个数字	0 或大于 6 个数字
数字串个数不满 6 个	左边补空格	" 123"左边是空格和字符混合
最高位与负号	两者紧邻"-12345"	最高位与负号之间有空格或字符

6.2.2 边界值分析

边界值分析是一种补充等价类划分法的测试用例设计方法,它不是选择等价类的任意元素,而是选择等价类边界的测试用例。实践证明,采用边界值分析法设计测试用例进行软件测试,常常可以查出更多的错误,取得良好的测试效果。因为,大量的错误发生在输入域或输出域的边界而不是其集合范围内。比如,在做三角形计算时,要输入三角形的三个边长:A、B 和 C。我们应注意到这三个数值满足:$A>0$、$B>0$、$C>0$、$A+B>C$、$A+C>B$、$B+C>A$,才能构成三角形。但如果把六个不等式中的任何一个大于号"$>$"错写成大于等于号"$\geq$",那就不能构成三角形。问题恰好出现在容易被疏忽的边界附近。

应用边界值分析法设计测试用例的原则就是确定输入或输出在边界取值,如:输入输出域的最大值或最小值、第一个值或最后一个值、最大个数或最小个数以及刚刚超出边界的值。

使用边界值分析方法设计测试方案首先应该确定边界情况,这需要经验和创造性,通常输入等价类和输出等价类的边界,就是应该着重测试的程序边界情况。选取的测试数据应该刚好等于、刚刚小于和刚刚大于边界值。下面是几条边界值分析方法选择测试用例的原则。

(1) 如果输入条件规定了值的范围,则应取刚达到这个范围的边界值,以及刚刚超越这个范围的边界值作为测试的输入数据。

例如,输入值的范围是"1~9",则可以选取"1"、"9"、"0.9"、"9.1"作为测试输入数据。

(2) 如果输入条件规定了输入值的个数,则用最大个数、最小个数、比最大个数大一个、比

最小个数小一个的数作为测试数据。

例如一个输入文件可以有 1～255 个记录，则可以分别设计有 1 个记录、255 个记录、0 个记录和 256 个记录的输入文件。

(3) 根据规格说明的每个输出条件，适用原则(1)。

例如，某程序的功能是计算折扣量，最低折扣是 0 元，最高折扣是 1000 元。则设计一些测试用例，使它们刚好产生 0 元和 1000 元的结果。

(4) 如果程序的规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。

(5) 分析规格说明，找出其他可能的边界条件。

通常设计测试用例时总是等价类划分和边界值分析两种技术同时使用，例如为了测试前面所述的把数字串转换为整数的程序，除了上一小节已用的等价类划分方法设计出的测试用例外，还应该用边界值分析法进行补充。

① 使输出刚好等于最小的负整数。

输入'－32768'；预期的输出－32768。

② 使输出刚好等于最大的正整数。

输入' 32767'；预期的输出 32767。

上一小节用等价类划分方法设计出来的测试用例(5)最好改为：

③ 使输出刚刚小于最小的负整数。

输入'－32769'；预期的输出："错误——无效的输入"。

原来的测试用例(6)最好改为：

④ 使输出刚刚大于最大的正整数。

输入' 32768'；预期的输出："错误——无效的输入"。

等价类划分法与边界值分析法相比，等价类划分法的测试数据是在各个等价类允许的值域内任意选取的，而边界值分析法的测试数据必须在边界值附近选取。一般来说，用边界值分析法设计的测试用例要比等价类划分法的代表性更广，发现错误的能力也更强。但边界值分析法对边界的分析与确定比较复杂，要求测试人员有更多的经验和耐心。

6.2.3　因果图

前面所介绍的等价类划分法和边界值分析法，都着重考虑输入条件，但未考虑输入条件之间的关系。因果图方法(Cause-Effect Graphics) 充分考虑了输入情况的各种组合及输入条件之间的相互制约关系。因而，该方法能够帮助我们按一定步骤、高效率地选择测试用例，同时还能指出程序规格说明描述中存在着什么问题。

用因果图生成测试用例的基本步骤如下所述。

(1) 分析软件规格说明描述中，哪些是原因(即输入条件或输入条件的等价类)，哪些是结果(即输出条件)，并给每个原因和结果赋予一个标识符。

(2) 分析软件规格说明书描述中的语义，找出原因与结果之间、原因与原因之间对应的关系，然后根据这些关系，画出因果图。

(3) 由于语法或环境限制，有些原因与原因之间、原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件。

(4) 把因果图转换成判定表。

(5) 把判定表的每一列拿出来作为依据，设计测试用例。

下面介绍一下因果图中出现的基本符号。

通常在因果图中用Ci表示原因，用Ei表示结果，其基本符号如图6-13所示。各节点表示状态，可取值“1”或“0”。“1”表示某状态出现，“0”表示某状态不出现。

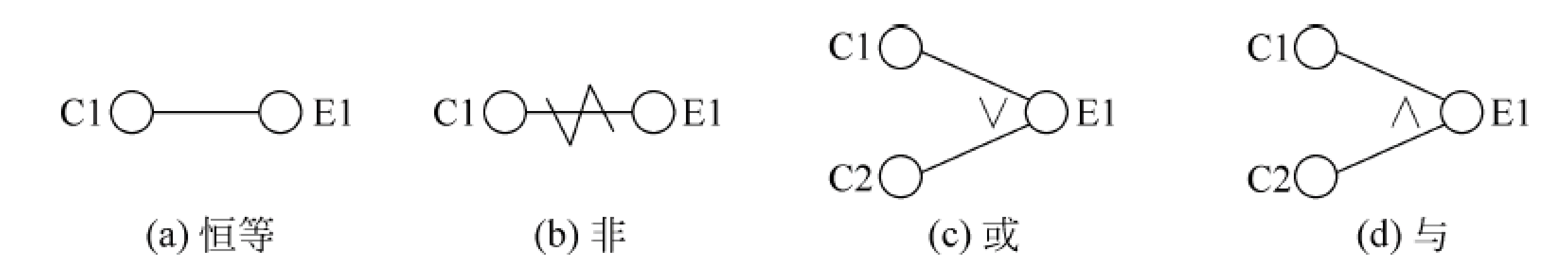


图6-13　因果图表示的基本符号

主要原因和结果之间的关系如下。

(a) 恒等：表示原因与结果之间一对一的对应关系。若原因出现，则结果出现；若原因不出现，则结果也不出现。

(b) 非：表示原因与结果之间的一种否定关系。若原因出现，则结果不出现；若原因不出现，反而结果出现。

(c) 或(V)：表示若几个原因中有一个出现，则结果出现，只有当这几个原因都不出现时，结果才不出现。

(d) 与(∧)：表示若几个原因都出现，结果才会出现。若几个原因中有一个不出现，结果就不会出现。

下面是因果图中表示约束条件的符号。

为了表示原因与原因之间、结果与结果之间有可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号。若从原因考虑，有以下4种约束，如图6-14所示。

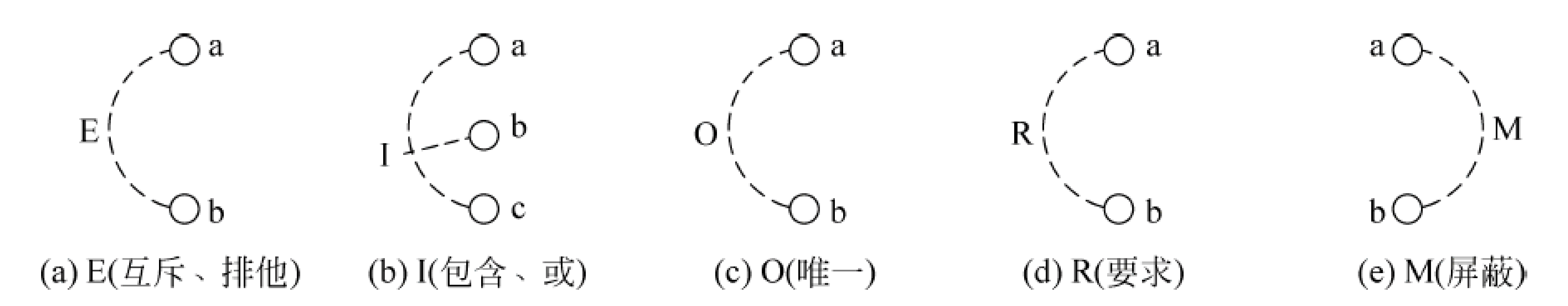


图6-14　因果图中表示约束条件的符号

(a) E(互斥)：表示a,b两个原因不会同时成立，两个中最多有一个可能成立。

(b) I(包含)：表示a,b,c三个原因中至少有一个必须成立。

(c) O(唯一)：表示a和b当中必须有一个，且仅有一个成立。

(d) R(要求)：表示当a出现时，b也必须出现。不可能a出现，而b不出现。

若从结果考虑，还有一种约束。

(e) M(屏蔽)：表示当a是1时，b必须是0。而a为0时，b的值不确定。

采用此方法时，首先分析规格说明书，确定原因和结果并把原因和结果连接成因果图。然后根据实际情况在因果图上使用若干特殊符号标明约束条件，最后把因果图转换成判定表，把判定表中每列的情况写成测试用例。

例如，有一个处理单价为5角钱的饮料的自动售货机软件测试用例的设计。其规格说明如下：若投入5角钱或1元钱的硬币，按下“橙汁”或“啤酒”的按钮，则相应的饮料就送出来；若售货机没有零钱找，则一个显示“零钱找完”的红灯亮，投入1元硬币并按下按钮后，饮料不送出来而且1元硬币也退出来；若有零钱找，则显示“零钱找完”的红灯灭，在送出饮料的同时退还5角硬币。

(1) 分析这一段说明，列出原因和结果。

原因：1. 售货机有零钱找；2. 投入 1 元硬币；3. 投入 5 角硬币；4. 按下“橙汁”按钮；5. 按下“啤酒”按钮。

结果：21. 售货机“零钱找完”灯亮；22. 退还 1 元硬币；23. 退还 5 角硬币；24. 送出橙汁饮料；25. 送出啤酒饮料。

建立中间节点，表示处理过程的中间状态：11. 投入 1 元硬币且按下“饮料”按钮；12. 按下“橙汁”或“啤酒”按钮；13. 应当找 5 角零钱并且售货机有零钱找；14. 钱已付清。

(2) 画出因果图，如图 6-15 所示。所有原因节点列在左边，所有结果节点列在右边。

(3) 由于 2 与 3、4 与 5 不能同时发生，所以分别加上约束条件 E。

(4) 因果图转换为判定表，如图 6-16 所示。判定表中没有被划去的一列就是一个测试用例。

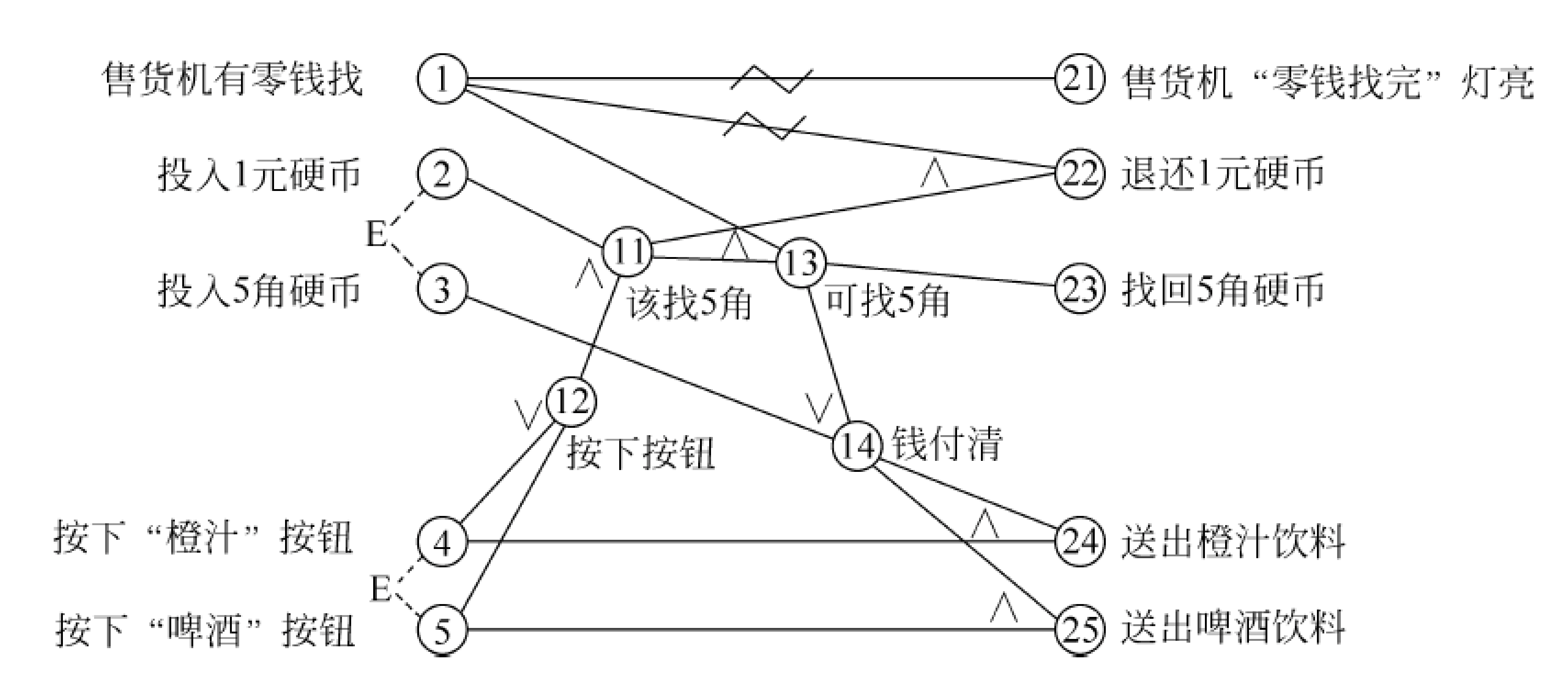


图 6-15　自动售货机因果图

序号		1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1	2
条件	①	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	②	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	③	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	④	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	⑤	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结果	⑪						1	1	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	⑫						1	1	0		1	1	0		1	1	0						1	1	0		1	1	0		1	1	0
	⑬						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	⑭						1	1	0		1	1	1		0	0	0						0	0	0		1	1	1		0	0	0
结果	㉑						0	0	0		0	0	0		0	0	0						1	1	1		1	1	1		1	1	1
	㉒						0	0	0		0	0	0		0	0	0						1	1	0		0	0	0		0	0	0
	㉓						1	1	0		0	0	0		0	0	0						0	0	0		0	0	0		0	0	0
	㉔						1	0	0		1	0	0		0	0	0						0	0	0		1	0	0		0	0	0
	㉕						0	1	0		0	1	0		0	0	0						0	0	0		0	1	0		0	0	0
测试用例							Y	Y	Y		Y	Y	Y		Y	Y							Y	Y	Y		Y	Y	Y		Y	Y	

图 6-16　对应自动售货机因果图的判定表

判定表中阴影部分表示因违反约束条件的不可能出现的情况，删去；第 16 列与第 32 列因什么动作也没做，也删去；最后可把剩下的 16 列作为确定测试用例的依据。

总结：因果图方法是一个非常有效的黑盒测试方法，它能够生成没有重复性的且发现错误能力强的测试用例，而且对输入、输出同时进行了分析；从因果图生成的测试用例包括了所有输入数据取 TRUE 与取 FALSE 的情况，构成的测试用例数目达到最少，且测试用例数目随输入数据数目的增加而线性地增加；如果哪个开发项目在设计阶段就采用了判定表，也就不必再画因果图，而是可以直接利用判定表设计测试用例了。

6.2.4 随机测试

随机测试指测试输入数据是所有可能输入值中随机选取的，是一种基本的"黑盒"测试方法。随机选取用随机模拟的方法，包括用伪随机数发生器、硬件随机模拟器产生输入数据。这种方法能够获得大量的测试数据，测试人员只需规定输入变量的取值区间、在需要的时候提供必要的变换机制，使产生的随机数服从预期的概率分布。

关于随机测试数据的选取方式，Laemmel 经分析认为，在测试次数很大时，可在数据输入空间按均匀分布选用，在测试次数较少时最好在常用的输入数据域以及最可能发生错误的输入数据域选用。随机测试与前面介绍的其他方法一起使用效果更佳。

但是随机测试不能事先将测试的输入数据存入文档，在排错时无法重现测试中错误发生的过程，也无法进行回归测试。补救的办法是将随机产生的测试数据记录下来备用。

6.2.5 猜错法

使用边界值分析法和等价类划分法，有助于设计出具有代表性的，容易暴露程序错误的测试方案。但是，不同类型不同特点的程序通常又有一些特殊的容易出错的情况。此外，有时分别使用每组测试用例时程序都能正常使用，这些输入数据的组合却有可能检验出程序中存在的错误。一般来说，即使是一个比较小的程序，可能的输入组合数也往往十分巨大，因此必须依靠测试人员的经验和直觉，从各种可能的测试方案中选出一些最有可能引起程序出错的方案。猜错法就是这样一种"黑盒"测试方法。

猜错法是基于经验和直觉推测程序中所有可能存在的各种错误，从而有针对性地设计测试用例的方法。猜错法的基本思想是：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。经验告诉我们，程序中容易出错的情况有：①输入数据为零或输出数据为零；②如果输入或输出的数目允许变化，则输入或输出的数目为 1 或 0 等。除此之外，还要仔细分析程序的规格说明书，注意找出其中遗漏或省略的部分，以便设计出相应的测试方案，检测程序员对这些部分的处理是否正确。

此外，经验表明，在一段程序中已经发现的错误数目往往和尚未发现的错误数目成正比。例如，在 IBM OS/370 操作系统中，用户发现的全部错误的 47%只与该系统 4%的模块有关。因此，在进一步测试时要着重测试那些已经发现较多错误的程序段。

6.3 "灰盒"测试

1999 年，美国洛克希德公司发表了"灰盒"测试法的论文，提出了"灰盒"测试法。"灰盒"测试是一种综合测试法，它将"黑盒"测试、"白盒"测试、回归测试和变异(Mutation)测试结合在一起，构成一种无缝测试技术。它是一种软件全生命周期测试方法，该方法通常是深入到用 Ada/C/Fortran 或汇编语言开发的嵌入式应用软件代码中进行功能的测试，或者与 Web 服务应用一起使用。

6.3.1 “灰盒”测试概念

“白盒”测试和“黑盒”测试各有其自身的特点，但也都存在着明显的不足，主要表现在只考虑了程序某一方面的属性和特征，缺少综合考虑。在“白盒”测试过程中，测试者可以看到被测程序的源代码，通过分析程序的内部结构，根据其内部结构设计测试用例。理想的“白盒”测试应该使选取的测试用例覆盖所有的路径，然而，这是不可能的，而且“白盒”测试它不关注被测程序的外部功能；“黑盒”测试是在测试者完全不考虑程序内部结构和内部特征（或对于上述信息无从获知）的情况下，根据需求规格说明书设计测试用例和推断测试结果的正确性。“黑盒”测试的不足在于，测试用例的选择只考虑了程序的输入，以及在该情况下的输出，并没有考虑程序的内部结构。因此，程序内部结构是否规范、结构化程度的好坏、系统的性能如何等都得不到测试。这样，要进行较全面的程序测试，不得不把测试工作用“白盒”测试方法和“黑盒”测试方法分别测试一次，这样做不但浪费时间，而且测试的效果不一定好。“灰盒”测试正是基于这一点提出的。

1. “灰盒”测试概述

“灰盒”测试是一种综合测试法，它将“黑盒”测试、“白盒”测试结合在一起，是基于程序运行时的外部表现同时又结合程序内部逻辑结构来设计用例，执行程序并采集程序路径执行信息和外部用户接口结果的测试技术。“灰盒”测试法的目的是验证软件满足外部指标以及软件的所有通道或路径都进行了检验。

“灰盒”测试以程序的主要功能和主要性能为测试依据，测试方法主要根据程序的流程图、功能说明书以及测试者的实践经验来设计。这里所说的主要功能和主要性能凭借测试者的经验来确定，即可把容易发生错误的变量域及程序流图作为测试的内容，而把那些不容易发生错误的变量输入和程序流图中不影响或不改变内部逻辑的细节忽略。事实上，许多测试工作是在不完全了解程序的内部逻辑时进行，这也就是“灰色”的由来。

同时，“灰盒”测试涉及输入和输出，但通常使用关于代码和程序操作等在测试人员视野之外的信息设计测试。在现在的测试工程中，最常见的“灰盒”测试是集成测试，重点关注软件系统的各个模块之间的相互关联，即模块之间的互相调用、数据传递、同步/互斥等。但是“灰盒”测试的概念已经由原来单一的“黑盒”测试和“白盒”测试的一些测试方法的简单叠加，衍生出许许多多新颖的分析方法。

2. “灰盒”测试特性

跟“黑盒”测试和“白盒”测试相比，“灰盒”测试有以下特性。

（1）“灰盒”测试同“黑盒”测试一样，也是根据需求规格说明文档来进行测试用例的设计。但它要深入到系统内部的特殊点来进行功能测试和结构测试，如进行单元接口测试，就是通过将接口参数传递到被测单元中，检验软件在测试执行环境控制下的执行情况。

（2）“灰盒”测试通常在程序员做完“白盒”测试之后，在功能测试人员进行大规模集成测试之前进行。

（3）“灰盒”测试需要了解代码工程的实现。

（4）“灰盒”测试是通过类似“白盒”测试的方法进行的，是通过编写代码，调用函数或者封装好的接口进行，但无须关心程序模块内部的实现细节，依然可把它当成一个黑盒。

（5）“灰盒”测试是由测试人员进行测试的。

在软件测试领域，对“灰盒”测试的应用属于比较新型的尝试，它打破了长久以来“黑盒”和“白盒”测试技术在这一领域的统治地位。DO—178B 规范也新进加入了利用“灰盒”测试方法

来进行测试的标准。

3. “灰盒”测试的优点

“灰盒”测试有以下优点：

(1) 能够进行基于需求的覆盖测试和基于程序路径覆盖的测试；

(2) 测试结果可以对应到程序内部路径，便于 bug 的定位、分析和解决；

(3) 能够保证设计的“黑盒”测试用例的完整性，防止遗漏软件的一些不常用的功能或功能组合；

(4) 能够避免需求或设计不详细或不完整对测试造成的影响。

4. “灰盒”测试的不足

“灰盒”测试也有如下的不足：

(1) 投入的时间比“黑盒”测试大概多 20%～40%的时间。

(2) 对测试人员的要求比“黑盒”测试高；“灰盒”测试要求测试人员清楚系统内部由哪些模块构成，模块之间如何协作。

(3) 不如“白盒”测试深入。

(4) 不适用于简单的系统。所谓的简单系统，就是简单到总共只有一个模块。由于“灰盒”测试关注系统内部模块之间的交互。如果某个系统简单到只有一个模块，那就没必要进行“灰盒”测试了。

6.3.2 “灰盒”测试步骤与应用举例

1. “灰盒”测试准备

“灰盒”测试前要做好充分的准备：

(1) 在测试中，除了部署产品外，还要安装源代码、编译源代码、运行软件。

(2) 需要代码覆盖率工具的配置；部署可以针对本软件开发语言的代码覆盖率工具。

(3) 测试人员要具备阅读代码的能力，其对开发语言的熟悉程度和对程序设计的经验多少决定了采用“灰盒”测试能够取得多大的好处，所以配置这方面的测试人员要进行必要的培训。

2. “灰盒”测试步骤

“灰盒”测试可采用 10 个步骤：①确定程序的所有输入和输出；②确定程序所有状态；③确定程序主路径；④确定程序的功能；⑤产生试验子功能 X 的输入，这里 X 为许多子功能之一；⑥制定验证子功能的 X 的输出；⑦执行测试用例 X 的软件；⑧检验测试用例 X 结果的正确性；⑨对其余子功能，重复第⑦和⑧步；⑩重复第④至⑧步，然后再进行第⑨步，进行回归测试。

程序功能正确系指在希望执行程序时，程序能够执行。子功能是指从进入子功能代码段到退出该代码段经过程序的一个路径。测试用例是由一组测试输入和相应的测试输出构成的。

3. “灰盒”测试实例

下面是根据一个实例来介绍一种传统的“白盒”测试与“黑盒”测试相结合的“灰盒”测试方法的应用。

1) 阅读需求

SDRD26537 (Software Design Requirement Document)

```
Requirement: Yes
Delivery: AESS
Magnetic Heading shall be ser invalid if value outside range of  - 180 inclusive and 180 exclusive.
[/TCAS TPA - 100X/Tests]
```

需求要求飞机在巡航过程中它的有效磁场角度范围为[－180,180]。(因为这是航空领域的实例,有些专业术语或缩写,但是它不影响阅读)

2) 分析需求

这个例子很简单,根据分析,测试人员优先选择“黑盒”测试方法的边界值分析方法,并确定取值范围为[－180,180]。设计一组边界值分析测试用例如下：－180.1，－180.0，－179.9，－1.0，0.0，1.0，180.0，179.9。

3) 根据分析写出测试用例脚本

详细的测试用例脚本由于篇幅太长,故不在这里一一写出。然后将测试用例脚本在测试环境里运行出结果。

但是在后面的测试工作中出现了意外,虽然测试用例的结果获得了通过,但是在做代码的“白盒”覆盖率时,未达到规定的覆盖率要求。为什么这么简单的一个单元测试失败了呢？在重新分析了需求和测试脚本以后,我们排除了这两方面带来的问题,原因很可能出在我们根据需求设计的脚本和源代码的实现有出入。

4) 分析相应的源代码

找到源代码的相应模块,如下所示：

```
//
//=========================================================================
const float MAX_VALID_ANGLE  =  180.0;
bool TcasAircraftInputSignallfcClass::getTrueHeading(int * argValue)
{
  static const float scalingFactor  =  16384.0 / 90.0;
  float roundFactor  = (((1.0 / 16384.0)/2.0) * 90.0);
  float temp;
  if (trueHeading -> get(&temp))
  {
      temp = (temp < MAX_VALID_ANGLE  - roundFactor ? temp : MAX_VALID_ANGLE  -  roundFactor);
      temp = (temp > +- MAX_VALID_ANGLE + roundFactor?temp :  - MAX_VALID_ANGLE + roundFactor);
      if (temp < 0)
      {
          roundFactor  =   - roundFactor;
      }
       * argValue  =  (int)((temp  +  roundFactor) * scalingFactor);
      return(true);
  }
  else
  {
      //return false signal is invalid
    return(false);
  }
}
```

经过对源代码的仔细分析,果然发现了问题所在。由于“黑盒”测试的特征以及 DO—178B 的规范,测试人员是完全根据需求文档来设计的测试用例。而需求文档在设计的时候设置的磁角度精确值统一为 0.1,但是在实际软件开发过程中,因为可靠性的要求,精确度提升到了 0.001。需求文档却未相应更新,导致最终的覆盖率失败。这里,不能取 179.9,而必须取

179.998，才能完全覆盖到语句，这就是“黑盒”测试与“白盒”测试相结合的产物。

另外很多常见的可用“灰盒”测试方法进行测试的例子，如：数据库中数据核查测试部分；协议中的消息头、消息体检查；终端里的中间模块的消息传递，比如AT指令的测试。

6.4 测试用例设计

Grenford J. Myers在*The Art of Software Testing*一书中提出：一个好的测试用例是指很可能找到迄今为止尚未发现的错误的测试，由此可见测试用例设计工作在整个测试过程中的地位。我们不能只凭借一些主观或直观的想法来设计测试用例，应该以一些比较成熟的测试用例设计方法为指导，再加上设计人员个人的经验积累来设计测试用例，两者相结合应该是非常完美的组合。

6.4.1 测试用例设计概念

测试用例目前没有经典的定义，比较通常的说法是：测试用例是为特定的目的而设计的一组测试输入、执行条件和预期的结果，体现测试方案、方法、技术和策略。内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。测试用例是执行的最小实体。简单地说，测试用例就是设计一个场景，使软件程序在这种场景下，必须能够正常运行并且达到程序所设计的执行结果。

随着中国软件业的日益壮大和逐步走向成熟，软件测试也在不断发展。其中，测试用例的设计和编制是软件测试活动中最重要的，它是测试工作的指导，是软件测试必须遵守的准则，更是软件测试质量稳定的根本保障。

1. 高质量测试用例应具备的特点

高质量测试用例应具备如下特点。

(1) 正确性。正确性是对测试用例最基本的要求，测试用例最好是要求输入用户实际数据以验证系统是否满足需求规格说明书的需求，并且测试用例中的测试点应保证至少覆盖需求规格说明书中的各项功能。

(2) 完整性。完整性同样是对测试用例最基本的要求，尤其是在一些基本功能项上，如有遗漏，那是不可原谅的。另外，完整性还体现在临界测试、压力测试、性能测试等方面，这方面测试用例也要能够涉及。

(3) 准确。按测试用例的输入实施测试后，要能够根据测试用例描述的输出得出正确的结论，不能出现模糊不清的语言。

(4) 清晰、简洁。好的测试用例描述清晰，每一步都应有响应，有很强的针对性，不应出现一些冗繁无用的操作步骤。测试用例不应太简单，也不能太过复杂，最大操作步骤最好控制在15步之内。

(5) 可维护性。由于软件开发过程中需求变更等原因的影响，常常需对测试用例进行修改、增加、删除等，以便测试用例符合相应测试要求。测试用例应具备这方面的功能。

(6) 适应性。测试用例应该适合特定的测试环境以及符合整个团队的测试水平，如纯英语环境下的测试用例最好使用英文编写。

(7) 可重用性。要求不同测试者在同样测试环境下使用同样测试用例都能得出相同结论。

(8) 其他。如可追溯性、可移植性也是对编写测试用例的一个要求。另外，好的测试用例

也是最有可能抓住错误的；还有，测试用例不要是重复的或多余的；最后，希望是一组相似测试用例中最有效的。

2. 测试用例设计基本原则

1）测试用例一般性设计原则

设计测试用例时，应遵循以下一般性原则。

（1）基于测试需求的原则。应按照测试类别的不同要求设计测试用例。如，单元测试依据详细设计说明，集成测试依据概要设计说明，配置项测试依据软件需求规格说明，系统测试依据用户需求（系统/子系统设计说明、软件开发计划等）。

（2）基于测试方法的原则。应明确测试用例的设计方法，为达到不同的测试充分性要求，应采用诸如等价类划分、边界值分析、猜错法、因果图等测试方法。

（3）兼顾测试充分性和效率的原则。测试用例应兼顾测试的充分性和测试的效率；测试用例的内容应完整，具有可操作性。

（4）测试用例的代表性。能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等。

（5）测试结果的可判定性。测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。

（6）测试执行的可再现性原则。即对同样的测试用例，系统的执行结果应当是相同的，也就是说是可再现的。

2）测试用例常用的设计原则

在测试用例一般性设计原则基础上，可提出更细或更具体的设计原则。

（1）一个测试用例对应一个功能点，每个测试用例都要有测试点。找准一个测试点则可，不要期望同时覆盖很多功能点，否则执行起来牵连太大，不易检查、维护和管理。

（2）测试用例要易读，要从执行者的角度去写测试用例，最好不要有太多的术语在里面，要指明具体位置。

（3）测试用例的执行粒度越小越好。因为只有这样，测试用例所覆盖的边界定义才会更加清晰，测试结果对产品问题的指向性更强，测试用例间的耦合度最低，彼此之间的干扰也就越低。这带来的直接好处是：测试用例的调试、分析和维护成本最低。

（4）步骤要清晰。一个测试用例有多个步骤，但要有重点。步骤指明人们怎么去操作。

（5）结果要明确。期望结果要清晰地指明一个操作之后应该看到什么结果（最好不要用正确、正常或者错误之类的含糊主观的字眼）。良好的测试用例一般会有两种状态：通过（Pass）、未通过（Failed）以及未进行测试（Not Done），如果测试未通过，一般会有测试的错误（Bug）报告进行关联；如未进行测试，则需要说明原因（测试用例本身的错误、测试用例目前不适用、环境因素等）。

（6）尽量将具有类似功能的测试用例抽象并归类。这主要是因为软件测试过程是无法进行穷举测试的，因此，对类似的测试用例的抽象过程显得尤为重要，一个好测试用例应该足能代表一组或者一系列的测试过程。

（7）尽量避免冗长和复杂的测试用例。为的是在测试执行过程中确保测试用例输出状态及验证结果的唯一性，从而便于跟踪和管理。

（8）总体设计思路是先进行基本功能测试，再进行复杂功能测试；先进行一般用户使用测试，再进行特殊用户使用测试；先进行正常情况的测试，再进行异常情况（内存和硬件的冲突、内存泄漏、破坏性测试等）的测试，这样可以先易后难，循序渐进，确保正常情况下基本功能

能够走通。

(9) 试着用测试用例替代产品文档。由于很多软件开发文档更新不及时,或者没有真正反映出产品的变化,这样也就无法准确地反映出产品功能当前的状态。而最终真正能够一直准确、有效地描述产品的功能的只有产品代码和测试用例。产品代码实现了产品功能,它一定是准确描述了产品的当前功能。另外,测试也应该忠实反映了产品功能,否则测试用例就会执行失败。因此,可把对测试用例的理解上升到一个新的高度,让它扮演产品描述文档的功能。这就要求我们编写的测试用例足够详细,测试用例的组织要有调理、分主次,并用测试用例管理工具来支撑。

(10) 测试用例设计要考虑单次投入成本和多次使用成本。因为编写测试用例一般是在测试的计划阶段进行的,虽然后期会有小的改动,但绝大多数是在一开始的设计阶段就基本上成型了;自动化测试用例也是如此,它也属于一次性投入;测试用例(包括手工和自动化测试用例)的执行则是多次投入成本,因为每一个新版本建立时都要执行所有的测试用例、分析测试结果、确定测试失败的原因,以验证该版本整体质量是否达到了指定的标准。

3. 测试用例覆盖内容

测试用例要测试被测软件所有的功能,并且希望选用少量、高效的测试数据进行尽可能完备的测试。测试用例应覆盖以下几个方面。

(1) 正确性测试。输入用户实际数据以验证系统满足需求规格说明书的要求;测试用例中的测试点应首先保证要至少覆盖需求规格说明书中的各项功能,并且正常。

(2) 容错性(健壮性)测试。程序能够接收正确的数据输入并且产生正确(预期)的输出,输入非法数据(非法类型、不符合要求的数据、溢出数据等),程序应能给出提示并进行相应处理。把自己想象成一名对产品操作一点也不懂的客户在进行任意操作。

(3) 完整(安全)性测试。对未经授权的人使用软件系统或数据的企图系统能够控制的程度,程序的数据处理能够保持外部信息(数据库或文件)的完整。

(4) 接口间测试。测试各个模块相互间的协调和通信情况、数据输入、输出的一致性和正确性。

(5) 数据库测试。依据数据库设计规范对软件系统的数据库结构、数据表及其之间的数据调用关系进行测试。

(6) 边界值分析法。确定边界情况(刚好等于、稍小于和稍大于和刚刚大于等价类边界值),针对被测系统在测试过程中主要在边界值附近选取输入的一些合法数据/非法数据。

(7) 压力测试。输入 10 条记录运行各个功能,输入 30 条记录运行,输入 50 条记录运行……进行测试。

(8) 等价类划分。将所有可能的输入数据(有效的和无效的)划分成若干个等价类。

(9) 错误推测。主要是根据测试经验和直觉,参照以往的软件系统出现错误之处。

(10) 效率。完成预定的功能,系统的运行时间(主要是针对数据库而言)。

(11) 可理解(操作)性。理解和使用该系统的难易程度(界面友好性)。

(12) 可移植性。在不同操作系统及硬件配置情况下的运行性。

(13) 回归测试。按照测试用例将所有的测试点测试完毕,测试中发现的问题开发人员已经解决,进行下一轮的测试。

(14) 比较测试。将已经发行的类似产品或原有的老产品与被测产品同时运行比较,或与已往的测试结果比较。

说明:针对不同的测试类型和测试阶段,测试用例编写的侧重点有所不同。

- 其中(1)、(2)、(6)、(8)、(9)、(13)为模块(组件、控件)测试、组合(集成)测试、系统测试都涉及的并要重点进行测试。
- 单元(模块)测试(组件、控件)测试要重点测试(5)。
- 组合(集成)测试重点进行接口间数据输入及逻辑的测试,即(4)。
- 系统测试重点测试(3)、(7)、(10)、(11)、(12)、(14)。
- 其中压力测试和可移植性测试如果是公司的系列产品,选用其中有代表性的产品进行一次代表性测试即可。
- 在基础的功能测试用例设计完成后,其他的测试项目只编写设计与之不同部分的测试用例。
- 每个测试项目的测试用例不是一成不变的,随着测试经验的积累或在测试其他项目发现有测试不充分的测试点时,可以不断地补充完善测试项目的测试用例。

6.4.2 测试用例编写要素与模板

测试用例设计是测试工作的核心任务之一,也是工作量最大的任务之一。编写良好的测试用例能够较好地提高测试用例的设计质量,方便跟踪测试用例的执行结果,自动生成测试用例的覆盖率报告。一般来说,编写测试用例所涉及的内容或要素,以及样式均大同小异,都包含主题、前置条件、执行步骤、期望结果等。测试用例可以用数据库、Word、Excel、Xml 等格式进行存储和管理,市面亦有成熟的商业软件工具和开源工具等。

1. 测试用例编写要素

一般测试用例应包括以下要素。

(1) 名称和标识。每个测试用例应有唯一的名称和标识符。

(2) 测试追踪。说明测试所依据的内容来源,如系统测试依据的是用户需求,配置项测试依据的是软件需求,集成测试和单元测试依据的是软件设计。

(3) 用例说明。简要描述测试的对象、目的和所采用的测试方法。

(4) 测试的初始化要求。应考虑一些初始化要求:①硬件配置(被测系统的硬件配置情况,包括硬件条件或电气状态);②软件配置(被测系统的软件配置情况,包括测试的初始条件);③测试配置(测试系统的配置情况,如用于测试的模拟系统和测试工具等的配置情况);④参数设置(测试开始前的设置,如标志、第一断点、指针、控制参数和初始化数据等的设置);⑤其他对于测试用例的特殊说明。

(5) 测试的输入。在测试用例执行中发送给被测对象的所有测试命令、数据和信号等。对于每个测试用例应提供如下内容:①每个测试输入的具体内容(如确定的数值、状态或信号等)及其性质(如有效值、无效值、边界值等);②测试输入的来源(如测试程序产生、磁盘文件、通过网络接收、人工键盘输入等),以及选择输入所使用的方法(如等价类划分、边界值分析、错误推测、因果图、功能图方法等);③测试输入是真实的还是模拟的;④测试输入的时间顺序或事件顺序。

(6) 期望的测试结果。说明测试用例执行中由被测软件所产生期望的测试结果,即经过验证,认为正确的结果。必要时,应提供中间的期望结果。期望测试结果应该有具体内容,如确定的数值、状态或信号等,不应是不确切的概念或笼统的描述。

(7) 评价测试结果的准则。判断测试用例执行中产生的中间和最后结果是否正确的准则。对于每个测试结果,应根据不同情况提供如下信息:①实际测试结果所需的精度;②实际测试结果与期望结果之间的差异允许的上限、下限;③时间的最大和最小间隔,或事件数目

的最大和最小值；④实际测试结果不确定时，再测试的条件；⑤与产生测试结果有关的出错处理；⑥上面没有提到的其他准则。

(8) 操作过程。实施测试用例的执行步骤。把测试的操作过程定义为一系列按照执行顺序排列的相对独立的步骤，对于每个操作应提供：①每一步所需的测试操作动作、测试程序的输入、设备操作等；②每一步期望的测试结果；③每一步的评价准则；④程序终止伴随的动作或出错指示；⑤获取和分析实际测试结果的过程。

(9) 前提和约束。在测试用例说明中施加的所有前提条件和约束条件，如果有特别限制、参数偏差或异常处理，应该标识出来，并要说明它们对测试用例的影响。

(10) 测试终止条件。说明测试正常终止和异常终止的条件。

2. 测试用例编写模板

在编写测试用例过程中，需要参考和规范一些基本的测试用例编写标准。

1) ANSI/IEEE829—1983 标准

在 ANSI/IEEE829—1983 标准中列出了和测试设计相关的测试用例编写规范和模板。标准模板中主要元素如下。

(1) 标识符(Identification)：每个测试用例应该有一个唯一的标识符，它将成为所有和测试用例相关的文档/表格引用和参考的基本元素，这些文档/表格包括设计规格说明书、测试日志表、测试报告等。

(2) 测试项(Test Item)：测试用例应该准确地描述所需要测试的项及其特征，测试项应该比测试设计说明书中所列出的特性描述更加具体，例如做 Windows 计算器应用程序的窗口设计，测试对象是整个应用程序的用户界面，这样测试项就应该是应用程序的界面特性要求，例如窗口缩放、界面布局、菜单等测试。

(3) 测试环境要求(Test Environment)：用来表征执行该测试用例需要的测试环境，一般来说，在整个的测试模块里面应该包含整个的测试环境的特殊要求，而单个测试用例的测试环境需要表征该测试用例所单独需要的特殊环境需求。

(4) 输入标准(Input Criteria)：用来执行测试用例的输入需求。这些输入可能包括数据、文件，或者操作(例如鼠标的左键单击、鼠标的按键处理等)，必要的时候，相关的数据库、文件也必须被罗列。

(5) 输出标准(Output Criteria)：标识按照指定的环境和输入标准得到的期望输出结果。如果可能的话，尽量提供适当的系统规格说明书来证明期望的结果。

(6) 测试用例之间的关联：用来标识该测试用例与其他的测试(或其他测试用例)之间的依赖关系，例如，用例 A 要求 B 的测试结果正确才能进行，此时需要在 A 的测试用例中表明对 B 的依赖性，从而保证测试用例的严谨性。

2) 某公司的测试用例编写规范

(1) 测试用例命名规则：以功能模块和业务流程进行命名。

(2) 测试用例编号规则：以测试模块名称的第一个字母进行命名(大写)，若测试模块名称比较长时，可进行简写。一般简拼不超过 5 个字母。如：测试模块为“用户管理”，功能编号为 YHGL；测试模块为“行政单位管理”，功能编号为 DWGL。功能编号也可以直接以 001、002、003、…来命名。

(3) 测试用例文档书写内容：①被测试对象介绍及测试范围与目的；②测试环境与测试辅助工具的描述；③功能测试用例主要元素；④前置条件(可选)及操作，如系统权限配置或前、后台配置描述(所有进行操作的前提条件)和测试操作步骤描述；⑤功能点，即功能点描

述；⑥输入数据，即前期数据准备；⑦预期结果，即描述输入数据后程序应该输出的结果；⑧测试结果，即描述本测试用例的实际测试情况，并判断实际测试结果与预期结果的差别；⑨Bug 编号/Bug 简要描述；⑩备注，即测试过程中遇到的问题等情况说明。

3）测试用例编写实例

以常见的 Web 登录页面测试为例(当用户没有输入用户名和密码时，不立即弹出错误对话框，而是在页面上使用红色字体来提示；用户密码使用掩码号“ * ”来标识；* 代表必选字段，将出现在输入文本框的后面；当登录出现错误时，在页面的顶部会出现相应的错误提示，错误提示是高亮的红色字体实现)，给出某公司所编写的测试用例，如表 6-3 所示。

表 6-3 Web 登录页面的测试用例

字段名称	描述
标识符	1100
测试项	站点用户登录功能测试
测试环境要求	(1) 用户 pass/pass 为有效登录用户，用户 pass1/pass 为无效登录用户，pass'jean/password 为有效登录用户 (2) 浏览器的 cookie 未被禁用
输入标准	(1) 输入正确的用户名和密码，单击“登录”按钮 (2) 输入错误的用户名和密码，单击“登录”按钮 (3) 不输入用户名和密码，单击“登录”按钮 (4) 输入正确的用户名并不输入密码，单击“登录”按钮 (5) 输入带特殊字符(/、'、"和 #，如 pass'jean)的用户名和密码，单击“登录”按钮 (6) 三次输入无效的用户名和密码，尝试登录 (7) 第一次登录成功后，重新打开浏览器登录，输入上次成功登录的用户名的第一个字符
输出标准	(1) 数据库中存在的用户(pass/pass，pass'jean/password)能正确登录 (2) 错误的或者无效用户登录失败，页面的顶部出现红色字体：“错误：用户名或密码输入错误” (3) 用户名为空时，页面顶部出现红色字体提示：“请输入用户名” (4) 当密码为空且用户名不为空时，页面顶部出现红色字体提示：“请输入密码” (5) 含特殊字符(‘、/、“”、#)的用户名，如数据库中有该记录，将能正确登录；如无该用户记录，将不能登录，校验过程和普通的字符相同，不能出现空白页面或者脚本错误 (6) 三次无效登录后，第四次尝试登录会出现提示信息“您已经三次尝试登录失败，请重新打开浏览器进行登录”，此后的登录过程将被禁止 (7) 自动完成功能将被禁止，查看浏览器的 cookie 信息，将不会出现上次登录的用户名和密码信息，第一次使用一个新账户登录时，浏览器将不会提示“是否记住密码以便下次使用”对话框 (8) 所有的密码均以 * 方式显示
测试用例间的关联	1101(有效密码测试)

3. 编写测试用例的注意事项

编写测试用例有很多注意事项，这里给出的是某公司编写测试用例的 4 个注意事项。

1）功能检查

功能检查主要检查：①功能是否齐全，如增加、删除、修改，查询条件是否合理，用户使用是否方便；②功能是否多余；③功能是否可以合并；④功能是否可以再细分；⑤软件流程与

实际业务流程是否一致；⑥软件流程能否顺利完成；⑦各个操作之间的逻辑关系是否清晰；⑧各个流程数据传递是否正确；⑨模块功能是否与需求分析及概要设计相符；⑩批量增加、批量修改，增加、修改等录入比较频繁的界面或录入数据量较多的界面，是否支持全键盘或全鼠标操作，并且使用通用的键实现数据字段的有序切换。

2）面向用户的考虑

面向用户的考虑主要考虑：①软件操作方便性和可用性，如：按键次数是否最少，并不以开发实现技术为由进行限制，而是以用户使用方便性和应用软件约定以及通常的快捷键来实现提出合理建议。另外，面对用户的操作是否简单易学。②智能化考虑。③提示信息是否模糊不清或有误导作用。错误信息是否有用户语言风格的出错后续处理建议提示。④要求用户进行的操作是否多余，能否由系统替代。系统升级后，用户能否不做任何操作自动进行所有升级的数据、环境等准备工作，包括删除缓存等动作。⑤能否记忆操作的初始环境，无须用户每次都进行初始化设置。⑥是否不经确认就对系统或数据进行重大修改。⑦能否及时反映或显示用户操作结果。⑧操作是否符合用户习惯，比如热键。⑨各种选项的可用及禁用是否及时合理。⑩某些相似的操作能否做成通用模块。

3）数据处理

数据处理包括三方面的内容：数据输入、数据处理以及数据输出。

（1）输入数据主要包括：边界值、大于边界值、小于边界值、最大个数、最大个数加1、最小个数、最小个数减1、空值或空表、极限值、0值、负数、非法字符、日期和时间控制、跨年度数据、数据格式，以及数据之间的关联性、逻辑性，数据范围、格式限制是否合乎日常情理，如年龄不应为负数，身份证位数必须为15或18位且与性别严格相关联，与生日可以有区别（考虑到阴历阳历的问题）但不相同时给予提示，私人电话号码的长度且国内电话只能有数字及短横线标识区号等。

（2）数据处理主要包括：处理速度、处理能力、数据处理正确率、计算方式及计算结果准确性（要考虑数字精度的取舍问题、汇总数据与分项数据累加的误差问题等）。

（3）输出结果主要包括：正确率、输出格式、预期结果、实际结果（金额数字的可能要验证小写、大写的一致性，大写可能要测试多种金额的大写，包括没有整数的情况下、没有小数的情况下、带整数和小数的情况下……）。

4）软件流程测试

软件流程测试主要考虑：反流程操作、反逻辑操作、重复操作、反业务流程操作以及违反流程的或打乱流程的或不按操作手册的乱操作。

6.4.3 测试用例的设计步骤

设计测试用例的时候，需要有清晰的测试思路，对要测试什么，按照什么顺序测试，覆盖哪些需求做到心中有数。测试用例编写者不仅要掌握软件测试的技术和流程，而且要对被测软件的设计、功能规格说明、用户使用场景以及程序/模块的结构都有比较透彻的理解。测试用例设计一般包括以下几个步骤。

1. 测试需求分析

从软件需求文档中，找出被测软件/模块的需求，通过自己的分析、理解，整理成为测试需求，清楚被测试对象具有哪些功能。

测试需求应该在软件需求基础上进行归纳、分类或细分，方便测试用例设计。测试用例中的测试集与测试需求的关系是多对一的，即一个或多个测试用例集对应一个测试需求。测试

需求的特点是：包含软件需求，具有可测试性。

2．业务流程分析

软件测试，不单纯是基于功能的“黑盒”测试，还需要对软件的内部处理逻辑进行测试。为了不遗漏测试点，需要清楚地了解软件产品的业务流程。建议在做复杂的测试用例设计前，先画出软件的业务流程。如果设计文档中已经有业务流程设计，可以从测试角度对现有流程进行补充。如果无法从设计中得到业务流程，测试工程师应通过阅读设计文档，与开发人员交流，最终画出业务流程图。业务流程图可以帮助理解软件的处理逻辑和数据流向，从而指导测试用例的设计。

从业务流程上，应得到以下信息：主流程是什么，条件备选流程是什么，数据流向是什么，以及关键的判断条件是什么。

3．测试用例设计

完成了测试需求分析和软件流程分析后，开始着手设计测试用例。测试用例设计的类型包括功能测试、边界测试、异常测试、性能测试、压力测试等。在测试用例设计中，除了功能测试用例外，应尽量考虑边界、异常、性能等情况，以便发现更多的隐藏问题。

“黑盒”测试的测试用例设计方法有等价类划分、边界值划分、因果图分析和错误猜测，“白盒”测试的测试用例设计方法有语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、多重条件覆盖。在这里主要讨论“黑盒”测试。在设计测试用例的时候可以使用软件测试用例设计方法，结合前面的需求分析和软件流程分析进行设计。

(1) 功能测试：测试某个功能是否满足需求的定义，功能是否正确、完备。适合的技术：由业务需求和设计说明导出的功能测试、等价类划分。

(2) 边界测试：对某个功能的边界情况进行测试。适合的技术：边界值划分。

(3) 异常测试：对某些功能来说，其边界情况无法简单地了解或某些操作不完全是正确的但又是可能发生的，类似这样的情况需要书写相关的异常测试。适合的技术：由业务需求和设计说明导出的特殊业务流程、错误猜测法、边界值分析、内部边界值测试。

(4) 性能测试：检查系统是否满足在需求中所规定达到的性能，性能主要包括了解程序的内外部性能因素。内部性能因素包括测试环境的配置、系统资源使用状况；外部因素包括响应时间、吞吐量等。适合的技术：业务需求和设计说明导出的测试。

(5) 压力测试：压力测试又称强度测试，主要是检查系统运行环境在极限情况下软件运行的能力，比如说给一个相当大的负荷或网络流量对应用软件进行兼容测试：测试软件产品在不同的平台、不同的工具、相同工具的不同版本下功能的兼容性。

4．测试用例评审

测试用例设计完成后，为了确认测试过程和方法是否正确，是否有遗漏的测试点，需要进行测试用例的评审。

测试用例评审一般是由测试主管安排，参加的人员包括测试用例设计者、测试主管、项目经理、开发工程师、其他相关开发测试工程师。测试用例评审完毕，测试工程师根据评审结果，对测试用例进行修改，并记录修改日志。

5．测试用例更新完善

测试用例编写完成之后需要不断完善，软件产品新增功能或更新需求后，测试用例必须配套修改更新；在测试过程中发现设计测试用例时考虑不周，需要对测试用例进行修改完善；在软件交付使用后客户反馈的软件缺陷，而缺陷又是因测试用例存在漏洞造成，也需要对测试用例进行完善。一般小的修改完善可在原测试用例文档上修改，但文档要有更改记录。软件

的版本升级更新，测试用例一般也应随之编制升级更新版本。测试用例是“活”的，在软件生命周期中不断更新与完善。

6.4.4 测试用例分级

测试用例级别表明该测试用例的重要程度。定义测试用例的级别，即是定义该测试用例在测试执行过程中的重要程度，包括优先执行。测试用例的重要性并不对应测试用例可能造成的后果，而是对应测试用例的基本程度，一个可能导致死机的测试用例未必是高级别的，因为其触发条件可能相当特别。

1. 测试用例的级别

测试用例列表中有一项内容是用例级别，如表 6-4 所示。

表 6-4 测试用例列表

测试项目	测试子项目	用例编号	用例级别	输入值	预期输出	实测结果	备　注
总　数		—				—	—

测试用例一般分为 4 级。

1）第 1 级：基本

(1) 该级别的测试用例涉及被测系统的基本功能，其测试用例的数量应受到控制。

(2) 该级别的划分依据是该测试用例执行失败会导致多项重要功能无法运行，如：表单维护中的增加功能、最平常的业务使用等可以认为是发生概率较高且经常使用的一些功能。

(3) 该级别的测试用例在每一轮版本测试中都必须执行。

2）第 2 级：重要

(1) 该级别的测试用例测试被测系统的重要功能，因此该级的测试用例数量较多。

(2) 该级别的划分依据主要是针对一些功能交互相关、各种应用场景、使用频率较高的正常功能测试用例。

(3) 在非回归的系统测试版本中基本上都需要进行验证，以保证系统所有的重要功能都是正常实现的。在测试过程中可以根据当前版本的具体情况决定是否在回归测试中全部执行或者部分执行这些测试用例。

3）第 3 级：一般

(1) 该级别的测试用例涉及系统的一般功能，因此该级别的用例数量也较多。

(2) 该级别的划分依据是针对使用频率较低于第 2 级的测试用例。例如：数值或数组使用的便捷情况、特殊字符、字符串超长、与外部交互消息失败、消息超时、事务完整性测试、可靠性测试等。

(3) 在非回归的系统测试版本中不一定都进行验证，而且在系统测试的中后期并不一定需要每个版本都进行测试。

4）第 4 级：特殊（如果没有可以不适用该级别）

(1) 该级别的测试用例一般非常少。

(2) 该级别的划分依据是该测试用例对应较特殊的预置条件和数据设置。虽然某些测试用例发现过较严重的错误，但是那些测试用例的触发条件非常特殊，仍然应该被置于第 4 级测试用例中。如界面规范化的测试也可归入第 4 级测试用例。在实际测试中使用频率非常低、

对用户可有可无的功能测试用例。

(3) 在版本测试中有某些正常原因(包括环境、人力、时间等),经过测试经理同意可以不进行测试。

2. 测试用例的优先级

定义测试用例的级别是为说明测试用例在测试执行时的优先程度以及测试用例的重要程度,而测试用例执行的优先程度首先取决于该测试用例所测试的用户需求点的紧急程度、使用频率以及重要程度,测试用例的级别定义首先要继承测试需求点的优先级别,那么就应先对测试需求进行优先级定义,而后再对需求点对应的一系列测试用例的优先级别进行定义。

在根据用户需求和需求分析文档提取测试需求时,必须要知道所有需求中,哪些是用户急需使用的部分,哪些是用户使用频繁的部分,哪些是系统最不能出现错误的部分,等等,那么这些部分就是我们测试级别最高的需求点。为此,在定义测试用例级别时,应该考虑:①继承测试需求的优先级别;②测试用例在用户实际使用中的使用概率及频率;③测试用例导致被测软件出错的概率;④测试用例导致系统发生错误后对系统的危害性。

当然,测试用例的级别不是一定下来就不变的,根据实际测试过程中的实际情况,是可以变化的。如:最初定义级别低的一个测试用例,由于在实际测试中,导致被测软件出错的概率比较高,那么就应当在本轮测试结束前提高该测试用例的级别。再如:最初定义级别高的一个测试用例,由于在实际测试中导致被测软件出错的触发条件或操作组合或用户使用频率非常不易达到,或者,由于用户需求变更,导致此需求点变为使用概率非常低的功能点了,那么我们就应当在本轮测试结束前降低该测试用例的级别,等等。

6.4.5　软件测试用例设计的误区

测试的目的是尽可能发现程序中存在的缺陷,测试活动本身也可以被看做一个项目,也需要在给定的资源条件下尽可能达到目标。目前国内大部分的软件公司在测试方面配备的资源是不足的,因此我们必须在测试计划阶段明确测试的目标,一切围绕测试的目标进行。而在测试设计阶段,在围绕测试目标的基础上进行测试用例的设计。软件测试用例是为了有效发现软件缺陷而编写的包含测试目的、测试步骤、期望测试结果的特定集合。正确认识和设计软件测试用例可以提高软件测试的有效性,便于测试质量的度量,增强测试过程的可管理性。

在实际软件项目测试过程中,由于对软件测试用例的作用和设计方法的理解不同,测试人员(特别是刚从事软件测试的新人)对软件测试用例存在不少错误的认识,给实际软件测试带来了负面影响,下面我们就对这些认识误区进行列举和分析,避免在今后的测试用例设计过程中犯类似的错误。

(1) 能发现到目前为止没有发现的缺陷的用例是好的用例。

测试本身是一种“V&V”的活动,测试需要确保程序做了它应该做的事情和程序没有做它不该做的事情。因此,作为测试实施依据的测试用例,必须要能完整覆盖测试需求,而不应该针对单个的测试用例去评判好坏。

(2) 测试输入数据设计方法等同于测试用例设计方法。

现在一些测试书籍和文章中讲到软件测试用例的设计方法,经常有这样的表述:测试用例的设计方法包括等价类、边界值、因果图、错误推测法、场景设计法等。这种表述是很片面的,这些方法只是软件功能测试用例设计中如何确定测试输入数据的方法,而不是测试用例设计的全部内容。

这种认识的不良影响可能会使不少人认为测试用例设计就是如何确定测试的输入数据,

从而掩盖了测试用例设计内容的丰富性和技术的复杂性。

无疑，对于软件功能测试和性能测试，确定测试的输入数据很重要，它决定了测试的有效性和测试的效率。但是，测试用例中输入数据的确定方法只是测试用例设计方法的一个子集，除了确定测试输入数据之外，测试用例的设计还包括如何根据测试需求、设计规格说明等文档确定测试用例的设计策略、设计用例的表示方法和组织管理形式等问题。

在设计测试用例时，需要综合考虑被测软件的功能、特性、组成元素、开发阶段(里程碑)、测试用例组织方法(是否采用测试用例的数据库管理)等内容。具体到设计每个测试用例而言，可以根据被测模块的最小目标，确定测试用例的测试目标；根据用户使用环境确定测试环境；根据被测软件的复杂程度和测试用例执行人员的技能确定测试用例的步骤；根据软件需求文档和设计规格说明确定期望的测试用例执行结果。

(3) 强调测试用例设计得越详细越好。

在确定测试用例设计目标时，一些项目管理人员强调测试用例"越详细越好"。具体表现在两个方面：尽可能设计足够多的设计用例，测试用例的数量越多越好；测试用例尽可能包括测试执行的详细步骤，达到"任何一个人都可以根据测试用例执行测试"，追求测试用例越详细越好。

软件测试是受到时间、人力和资金等资源的约束。而这种做法和观点最大的危害就是耗费了很多的测试用例设计时间和资源，可能等到测试用例设计、评审完成后，留给实际执行测试的时间所剩无几了。因为当前软件公司的项目团队在规划测试阶段，分配给测试的时间和人力资源是有限的，而软件项目的成功要坚持"质量、时间、成本"的最佳平衡，没有足够多的测试执行时间，就无法发现足够多的软件缺陷，测试质量就无从谈起了。

编写测试用例的根本目的是有效地找出软件可能存在的缺陷，为了达到这个目的，需要分析被测试软件的特征，运用有效的测试用例设计方法，尽量使用较少的测试用例，同时满足合理的测试需求覆盖，从而达到"少花时间多办事"的效果。

测试用例中的测试步骤需要详细到什么程度，主要取决于测试用例的"最终用户"(即执行这些测试用例的人员)，以及测试用例执行人员的技能和产品熟悉程度。在测试计划阶段，一般给予测试设计30％～40％的时间，测试设计工程师能够根据项目的需要自行确定用例的详细程度，在测试用例的评审阶段由参与评审的相关人对其把关。总之，文档的作用主要用于沟通，只要能达到沟通的目的就可以了。

(4) 追求测试用例设计"一步到位"。

现在软件公司都意识到了测试用例设计的重要性了，但是一些人认为设计测试用例是一次性投入，测试用例设计一次就"万事大吉"了，片面追求测试设计的"一步到位"。

这种认识造成的危害性是使设计出的测试用例缺乏实用性，或者误导测试用例执行人员，误报很多不是软件缺陷的"Bug"，这样的测试用例在测试执行过程中"形同虚设"，难免沦为"垃圾文档"。

"唯一不变的是变化"。任何软件项目的开发过程都处于不断变化过程中，用户可能对软件的功能提出新需求，设计规格说明相应地更新，软件代码不断细化。设计软件测试用例与软件开发设计并行进行，必须根据软件设计的变化，对软件测试用例进行内容的调整、数量的增减，增加一些针对软件新增功能的测试用例，删除一些不再适用的测试用例，修改那些模块代码更新了的测试用例。

软件测试用例设计只是测试用例管理的一个过程，除此之外，还要对其进行评审、更新、维护，以便提高测试用例的"新鲜度"，保证"可用性"。因此，软件测试用例也要坚持"与时俱进"

的原则。

(5) 测试用例不应该包含实际的数据。

测试用例是"一组输入、执行条件、预期结果",因此测试用例毫无疑问地应该包括清晰的输入数据和预期输出,没有测试数据的用例最多只具有指导性的意义,不具有可执行性。当然,测试用例中包含输入数据会带来维护、与测试环境同步之类的变化问题。

(6) 测试用例中不需要明显的验证手段。

"预期输出"仅描述为程序的可见行为,其实,"预期结果"的含义并不只是程序的可见行为。例如,对一个订货系统,输入订货数据,单击"确定"按钮后,系统提示"订货成功",这样是不是一个完整的用例呢?是不是系统输出的"订货成功"就应该作为我们唯一的验证手段呢?显然不是。订货是否成功还需要查看相应的数据记录是否更新,因此,在这样的一个用例中,还应该包含对测试结果的显式的验证手段:在数据库中执行查询语句进行查询,看查询结果是否与预期的一致。

(7) 让测试新人设计测试用例。

刚参加测试工作的测试新手经常思考和询问的一个问题是:"怎么才能设计好测试用例?"因为他(她)们之前从来没有设计过测试用例,面对大型的被测试软件感到"老虎吃天,无从下口"。

让测试新手设计测试用例是一种高风险的测试组织方式,它带来的不利后果是设计出的测试用例对软件功能和特性的测试覆盖性不高,编写效率低,审查和修改时间长,可重用性差。而事实上,软件测试用例设计是软件测试的中、高级技能,不是每个人(尤其是测试新人)都可以编写的,测试用例编写者不仅要掌握软件测试的技术和流程,而且要对被测软件的设计、功能规格说明、用户使用场景以及程序/模块的结构都有比较透彻的理解。

因此,实际测试过程中,通常安排经验丰富的测试人员进行测试用例设计,测试新人可以从执行测试用例开始,随着项目进度的不断进展,测试人员的测试技术和对被测软件的不断熟悉,可以积累测试用例的设计经验,编写测试用例。

6.5 单元测试

软件单元是指软件设计说明中一个可独立测试的元素,是程序中一个逻辑上独立的部分,它不能再分解为其他软件成分,如软件源代码中单个的函数、源文件或类。

单元测试(模块测试)是开发者编写的一小段代码,用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言,一个单元测试是用于判断某个特定条件(或者场景)下某个特定函数的行为,是对单个的软件单元或者一组相关的软件单元进行的代码级别上的测试。

按照软件生命周期对软件测试所进行的级别划分,单元测试是最初始级别的测试,然后是集成测试(组装测试)、确认测试(配置项测试)和系统测试,如图 6-17 所示。

开始是单元测试,集中对用源代码实现的每一个程序单元进行测试,检查各个程序模块是否正确地实现了程序规定的功能。然后再把已经测试过的程序模块组装起来,进行集成测试,主要对与设计相关的软件体系结构的构造进行测试。在这里,将一个一个经过单元测试并确保无误的程序模块组装成软件系统,对其正确性和程序结构方面进行检查。确认测试则是要检查已经组装好的软件系统是否满足需求规格说明中明确说明了的各种需求,以及软件配置是否安全、正确。最后是系统测试,把经过确认测试的软件在实际环境中进行运行,并与其他系统组合在一起进行测试。

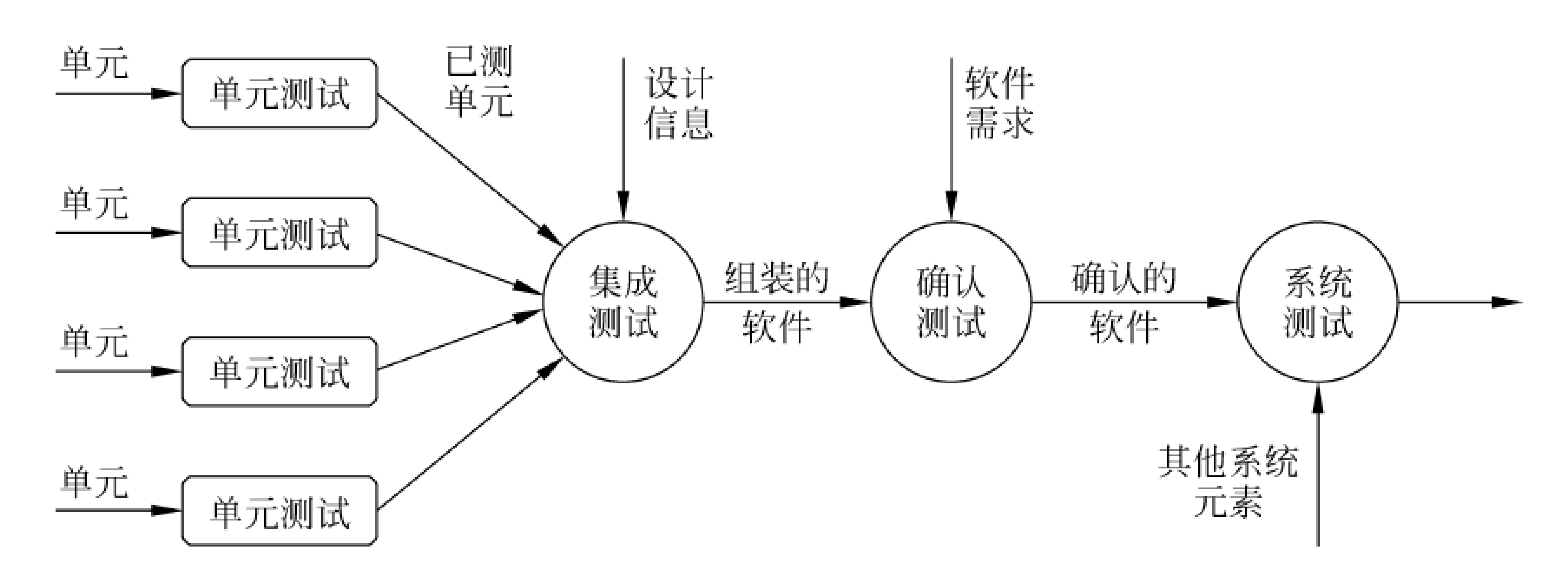


图 6-17 软件测试步骤

除对应于软件开发过程逐步进行验证和确认之外,软件测试分步骤进行的另一个意义在于,不同阶段的测试对应了不同层次的软件阶段产品,可以针对测试对象不同的特点采用不同的技术,发现不同特征的错误,使软件在多层次的测试中不断提高质量。

单元测试又称模块测试,是针对软件设计的最小单位——程序模块或功能模块,进行正确性检验的测试工作。其目的在于检验程序各模块中是否存在各种差错,是否能正确地实现其功能,满足其性能和接口要求。单元测试如果认真实施的话,效果会非常好。但是实施阻力比较大(主要是人员和管理因素),因此很多开发团队一般只在关键的程序单元中实施。单元测试有比较系统的理论和方法,但也依赖于系统的特殊性和开发人员的经验。单元测试有大量的辅助工具,开发人员也经常自己开发测试代码和测试工具。

单元测试验证程序和详细设计说明的一致性,需要从程序的内部结构出发设计测试用例,多个模块之间可以平行独立地进行单元测试。

程序模块是可由汇编程序、编译程序、装入程序或翻译程序作为一个整体来处理的一级独立的、可识别的程序指令,是大型程序指令的一个组成部分;功能模块指实现了一个完整功能的程序(单元),一个完整的程序单元具备输入、加工和输出三个环节,每个程序单元都应该有正规的规格说明,要对其输入、加工和输出的关系做出明确的描述。

6.5.1 单元测试的意义

1. 对单元测试的误解

对于单元测试,人们往往存在很多的误解。

(1) 浪费的时间太多。

一旦编码完成,缺乏软件工程实践经验的开发人员就会迫不及待地进行软件集成工作,这样就能看到实际系统开始启动工作,在这种开发步骤中,实际上的进步被表面上的进步所取代。现实中发现编码阶段引入的缺陷远远多于其他阶段,系统测试发现的缺陷大多数是编码缺陷。这样,系统能进行正常工作的可能性很小,更多的情况是充满了各式各样的 Bug。这些 Bug 包含在独立的单元里,其本身也许是琐碎、微不足道的,但在软件集成为一个系统时会增加额外的工期和费用。其实进行完整的单元测试和编写代码所花费的精力大致上是相同的,一旦完成了单元测试,确保手头拥有稳定可靠部件的情况下,再进行高效的软件集成才是真正意义上的进步。

程序的可靠性对软件产品的质量有很大的影响,在大型软件公司,每写一行程序,都可能要测试很多遍。由此可见大型软件公司对测试的重视程度。

(2) 软件开发人员不应参与单元测试。

目前很多开发团队的开发人员承担着包括设计、编码及测试多个角色(如参与或部分参与

高层设计,承担低层设计或程序实现,担当低层测试等)。他们强调开发人员做测试时间紧、效果不好,而且也不知怎么做,等等。但这些都不是理由,因为单元测试常常和编码同步进行,每完成一个模块就应进行单元测试。在对每个模块进行单元测试时,不能忽略和其他模块的关系,为模拟这一关系,需要辅助模块,因此若单独的测试人员进行单元测试,往往工作量大,周期长,耗费巨大,其结果事倍功半。单元测试是由程序员自己来完成,最终受益的也是程序员自己。可以这么说,程序员有责任编写功能代码,同时也就有责任为自己的代码编写单元测试,以保证它们实现了设计的功能(其实在很多情况下,开发者也应进行集成测试)。另外,对于程序员来说,如果养成了对自己写的代码进行单元测试的习惯,不但可以写出高质量的代码,而且还能提高编程水平。

(3) 它仅仅是证明这些代码做了什么。

这是那些没有首先为每个单元编写一个详细的规格说明而直接跳到编码阶段的开发人员提出的一条普遍的抱怨,编码完成以后并且面临代码测试任务的时候,他们就阅读这些代码并找出它实际上做了什么,把他们的测试工作基于已经写好的代码的基础上。当然,他们无法证明任何事情。所有的这些测试工作能够表明的事情就是编译通过。是的,他们也许能够抓住(希望能够)罕见的编译错误,但是他们能够做的仅仅是这些。

如果他们首先写好一个详细的规格说明,测试能够以规格说明为基础。这样就能够针对代码的规格说明,而不是针对自身进行测试。这样的测试仍然能够抓住编译错误,同时也能找到更多的编码错误,甚至是一些规格说明中的错误。好的规格说明可以使测试的质量更高,所以最后的结论是高质量的测试需要高质量的规格说明。

(4) 我是很棒的程序员,不需要进行单元测试。

传统的开发观念是:开发人员的任务是完成编程,让系统正确运行起来。如果程序有错就进行调试,程序通过调试任务就完成了。而且我是编程高手,自信自己的程序不会出错。

如果我们真正擅长编程并且有绝招,就应当不会有错误,但这只是一个神话。要记住的是:开发人员的任务是完成程序,直到交付和维护;另外,人的失误是不可避免的,无论多小心,都会有错误。因此,程序必须经过各种各样测试,单元测试只是其中一种。

(5) 不管怎样,集成测试或系统测试将会抓住所有的 Bug。

集成测试的目标是把通过单元测试的模块拿来,构造一个在设计中所描述的程序结构,通过测试发现和接口有关的问题。我们在测试工作开展的过程中,发现并提交进行合格性测试的软件,在测试过程中有很多 Bug,有些严重问题,甚至导致死机,以至于不能再测试其他功能,进行错误修改,回归测试时又发现其他新的问题,使得测试工作很难开展下去。

如果把单元测试的任务堆积到系统测试阶段,致使大量的故障堆积在项目中后期(项目后10%的工作,占用了项目90%的时间),造成故障难以定位或飘忽不定,开发、测试人员疲于奔命,费用成倍上升。

(6) 单元测试效率不高。

在实际工作中,开发人员不想进行单元测试,认为没有必要且效率不高,其实错误发生和被发现之间的时间与发现和改正该错误的成本是指数关系,频繁的单元测试能使开发人员排错的范围缩得很小,大大节约排错所需的时间,同时错误尽可能早地被发现和消灭会减少由于错误而引起的连锁反应。

在某一功能点上进行准备测试、执行测试和修改缺陷的时间,单元测试的效率大约是集成测试的两倍、系统测试的三倍。

通过对这些误解的分析,我们对于单元测试有了一个基本的了解,其实作为软件系统的最

小组成单位,单元测试具有这些属性:①它是由一个程序员完成的;②它有一个详细的设计说明,包括输入定义、输出定义和加工说明;③它是一个可识别的、看得见的程序组成部分,并容易被组合成程序;④能被单独地运行和测试;⑤它的规模比较小,逻辑比较简单。

2. 单元测试的好处

单元测试具有如下好处。

(1) 单元测试将注意力集中在程序的基本组成部分。首先保证每个单元测试通过,才能使下一步把单元组装成部件并测试其正确性有基础。单元是整个软件的构成基础,像硬件系统中的零部件一样,只有保证零部件的质量,这个设备的质量才有基础,单元的质量也是整个软件质量的基础。因此,单元测试的效果会直接影响软件的后期测试,最终在很大程度上影响到产品的质量。

(2) 单元测试可以平行开展。这样可以使多人同时测试多个单元,提高了测试的效率。

(3) 单元规模较小,复杂性较低。因而发现错误后容易隔离和定位,有利于调试工作。另外,也使单元测试可以使用包括"白盒"测试在内的许多测试技术,能够进行比较充分细致的测试,确保整个程序测试满足一些基本覆盖要求。

(4) 单元测试的测试效果是最显而易见的。做好单元测试,不仅后期的系统集成联调或集成测试和系统测试会很顺利,节约很多时间;而且在单元测试过程中能发现一些很深层次的问题,同时还会发现一些很容易发现而在集成测试和系统测试很难发现的问题;更重要的是单元测试不仅仅是证明这些代码做了什么,而是代码是如何做的,是否做了它该做的事情而没有做不该做的事情。

(5) 单元测试的好与坏直接关系到测试成本,也会直接影响到产品质量。因为单元测试中易发现的问题拖到后期测试发现,其成本将成倍数上升;另外,可能就是由于代码中的某一个小错误就导致了整个产品的质量降低一个指标,或者导致更严重的后果。

单元测试要求尽早地、可重复地、尽可能采用自动化的手段进行。事实上,单元测试是一种验证行为——测试和验证程序中每一项功能的正确性,为以后的开发提供支持;单元测试是一种设计行为——编写单元测试将使我们从调用者观察、思考,特别是在单元设计时要先考虑测试,这样就可把程序设计成易于调用和可测试的,并努力降低软件中的耦合,还可以使编码人员在编码时产生预测试,将程序的缺陷降低到最小;单元测试是一种编写文档的行为——是展示函数或类如何使用的最佳文档;单元测试具有回归性——自动化的单元测试有助于回归测试的开展。

6.5.2 单元测试的内容

单元测试是由一组独立的测试构成,每个测试针对软件中的一个单独的程序单元。单元测试并非检查程序单元之间是否能够合作良好,而是检查单个程序单元行为是否正确。在单元测试时,测试人员根据详细设计说明书和源程序清单,了解到该模块的I/O条件和模块的逻辑结构,主要采用"白盒"测试的测试用例,辅之以"黑盒"测试的测试用例,使之对任何合理和不合理的输入都要能鉴别和响应。这就要求对程序所有的局部和全局的数据结构、外部接口和程序代码的关键部分进行桌面检查和代码审查。

对被测模块或单元进行单元测试主要有5个方面的内容,如图6-18所示。

1. 单元测试5个方面的内容

1) 模块接口测试

在单元测试开始时,应该对通过所有被测模块的数据流进行测试。如果数据不能正常地

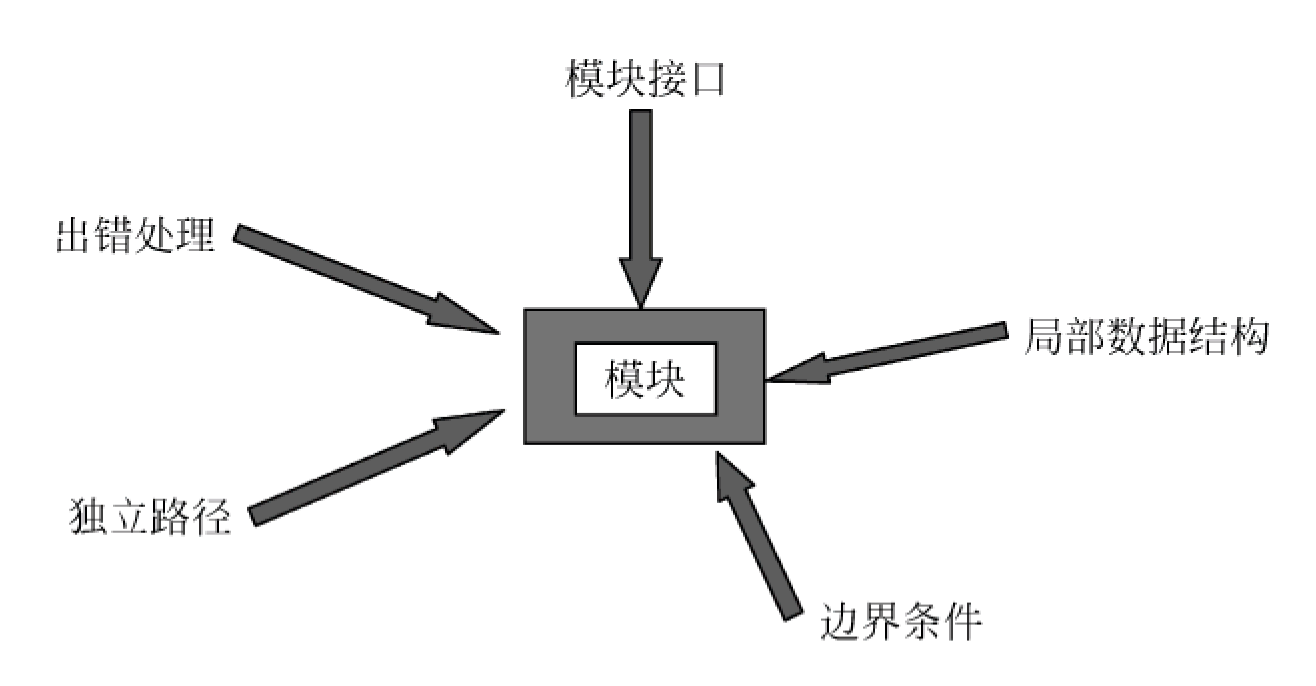


图 6-18 单元测试的内容

进入及输出,那么其他的全部测试都说明不了问题。Myers 在他关于软件测试的书中为接口测试提出了一个检查表:

(1) 模块输入参数的数目是否与模块形式参数数目相同;
(2) 模块各输入的参数属性与对应的形参属性是否一致;
(3) 模块各输入的参数类型与对应的形参类型是否一致;
(4) 传到被调用模块的实际参数的数目是否与被调用模块形式参数的数目相同;
(5) 传到被调用模块的实际参数的属性是否与被调用模块形式参数的属性相同;
(6) 传到被调用模块的实际参数的类型是否与被调用模块形式参数的类型相同;
(7) 引用内部函数时,实参的次序和数目是否正确;
(8) 是否引用了与当前入口无关的参数;
(9) 用于输入的变量值有没有改变;
(10) 在经过不同模块时,全局变量的定义是否一致;
(11) 限制条件是否以形参的形式传递;
(12) 使用外部资源时,是否检查可用性并及时释放资源,如内存、文件、硬盘、端口等。

当模块通过外部设备进行输入/输出操作时,必须扩展接口测试,附加如下的测试项目:

(1) 文件的属性是否正确;
(2) Open 与 Close 语句是否正确;
(3) 规定的格式是否与 I/O 语句相符;
(4) 缓冲区的大小与记录的大小是否相配合;
(5) 在使用文件前,文件是否打开;
(6) 文件结束的条件是否安排好了;
(7) I/O 错误是否检查并做了处理;
(8) 在输出信息中是否有文字错误。

2) 局部数据结构测试

模块的局部数据结构是最常见的错误来源,应设计测试用例以检查以下各种错误:

(1) 不正确或不一致的数据类型说明;
(2) 使用尚未赋值或尚未初始化的变量;
(3) 错误的初始值或错误的默认值;
(4) 变量名拼写错或书写错——使用了外部变量或函数;
(5) 不一致的数据类型;
(6) 全局数据对模块的影响;

(7) 数组越界；

(8) 非法指针。

3) 路径测试

检查由于计算错误、判定错误、控制流错误导致的程序错误。由于在测试时不可能做到穷举测试，所以在单元测试时要根据“白盒”和“黑盒”测试用例设计方法设计测试用例，对模块中重要的执行路径进行测试。重要的执行路径指那些处在完成单元功能的算法、控制、数据处理等重要位置的执行路径，也指由于控制较复杂而易错的路径，有选择地对执行路径进行测试是一项重要的任务。应当设计测试用例查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误，对基本执行路径和循环进行测试可发现大量的路径错误。

在路径测试中，要检查的错误有死代码、错误的计算优先级、算法错误、混用不同类的操作、初始化不正确、精度错误、比较运算错误、赋值错误、表达式的不正确符号(如＞、＞＝、＝、＝＝、！＝)、循环变量的使用错误(错误赋值)以及其他错误等。

比较操作和控制流向紧密相关，测试用例设计需要注意发现比较操作的错误：

(1) 不同数据类型的比较。

(2) 不正确的逻辑运算符或优先次序。

(3) 因浮点运算精度问题而造成的两值比较不等。

(4) 关系表达式中不正确的变量和比较符。

(5) “差 1 错”，即不正常的或不存在的循环中的条件。

(6) 当遇到发散的循环时无法跳出循环。

(7) 当遇到发散的迭代时不能终止循环。

(8) 错误地修改循环变量。

4) 错误处理测试

错误处理路径是可能引发错误处理的路径及进行错误处理的路径，错误出现时错误处理程序重新安排执行路线，或通知用户处理，或干脆停止执行使程序进入一种安全等待状态。测试人员应意识到，每一行程序代码都是可能执行到，不能自己认为错误发生的概率很小而不去进行测试。一般软件错误处理测试应考虑下面几种可能的错误：

(1) 出错的描述是否难以理解，是否能够对错误定位。

(2) 显示的错误与实际的错误是否相符。

(3) 对错误条件的处理正确与否。

(4) 在对错误进行处理之前，错误条件是否已经引起系统的干预等。

在进行错误处理测试时，我们要检查如下内容：

(1) 在资源使用前后或其他模块使用前后，程序是否进行错误出现检查。

(2) 出现错误后，是否可以进行错误处理，如引发错误、通知用户、进行记录。

(3) 在系统干预前，错误处理是否有效，报告和记录的错误是否真实详细。

5) 边界测试

边界测试是单元测试中最后的任务。软件常常在边界上错误。例如，在一个程序段中有一个 n 次循环，当到达第 n 次循环时就可能会出错；或者在一个有 n 个元素的数组中，第 n 个元素是很容易出错的。因此，要特别注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。

此外，如果对模块性能有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响运行时间的因素。下面是边界测试具体要检查的内容：

(1) 普通合法数据是否正确处理。

(2) 普通非法数据是否正确处理。

(3) 边界内最接近边界的(合法)数据是否正确处理。

(4) 边界外最接近边界的(非法)数据是否正确处理等。

(5) 在 n 次循环的第 0 次、第 1 次、第 n 次是否有错误。

(6) 运算或判断中取最大、最小值时是否有错误。

(7) 数据流、控制流中刚好等于、大于、小于确定的比较值时是否出现错误。

2. 单元测试要求

为了使单元测试能充分细致地展开,应在实施单元测试中遵守下述要求。

(1) 语句覆盖达到 100%。语句覆盖指被测单元中每条可执行语句都被一个测试用例所覆盖。语句覆盖是强度最低的覆盖要求,要考虑语句覆盖的意义,只要想象一下用一段从没执行过的程序控制庞大的飞行器升上天空,然后设法使它精确入轨,这种轻率简直就是荒唐。实际测试中,不一定能做到每条语句都执行到。第一,存在“死代码”,即由于程序设计错误在任何情况下都不可能执行到的代码。第二,不是“死代码”,但是由于要求的测试输入及条件非常难达到或单元测试的条件所限,使得代码没有得到运行。因此,在可执行语句未得到执行时,要深入程序作详细的分析。如果是属于以上两种情况,则可以认为完成了覆盖,但是对于后者,如果可能一定要尽量测试到,如果以上两者都不是,则是因为测试用例设计不充分,需要再设计测试用例。

(2) 分支覆盖达到 100%。分支覆盖指分支语句取真值和取假值各一次,分支语句是程序控制流的重要处理语句,在不同流向上测试可以验证这些控制流向的正确性。分支覆盖使这些分支产生的输出都得到验证,提高测试的充分性。

(3) 错误处理路径达到 100%。覆盖所有的错误处理路径。

(4) 单元的软件特性覆盖。软件的特性包括功能、性能、属性、设计约束、状态数目、分支的行数等。

(5) 各种数据特性覆盖。对使用额定数据值、奇异数据值和边界值的计算进行检验,用假想的数据类型和数据值运行,测试排斥不规则输入的能力。

单元测试通常是由编写程序的人自己完成的,但是项目负责人应当关心测试的结果。所有的测试用例和测试结果都是模块开发的重要资料,需妥善保存。

6.5.3 单元测试方法和步骤

在软件开发过程中,代码编写和单元测试共属实现阶段,编码完成并编译通过后才开始进行单元测试。

在进行动态的单元测试前要先对程序进行静态分析和代码审查。这是因为:

(1) 使用动态测试技术要准备测试用例,进行结果记录和分析,工作量大,如果错误太多会降低动态测试效率。

(2) 目前的动态测试技术局限性比较大,有相当类型的错误靠动态测试是难以发现的。因此,先使用静态分析和代码审查技术,能充分地发挥人的判断和思维优势,检查出对机器而言很难发现的错误。典型的包括代码和设计规格的一致性、代码逻辑表达式的正确性。这些检查在动态测试阶段将会是非常烦琐而又非常困难的。

(3) 有些错误在动态测试时无法检查到。

(4) 使用代码审查技术,一旦发现错误,就知道错误的性质和位置,调试代价较低。

(5) 使用静态分析方法一次就能揭示一批错误,并且随后就可以立即纠正错误。

由于单元测试针对程序单元,而程序单元并不是一个独立可运行的程序,因此,在考虑被测模块时,同时要考虑到它和外界其他模块的联系。

1. 单元测试方法

在考虑被测单元和外界其他模块的联系时,可用一些辅助模块去模拟与被测模块关联的其他模块。这些模块分为两种。

(1) 驱动模块。相当于被测模块的主程序。它接收测试数据,把这些测试数据传送给被测模块,最后再输出实测结果。

(2) 桩模块。由被测模块调用,用以代替由被测单元所调用的模块的功能,返回适当的数据或进行适当的操作,使被测单元能继续运行下去,同时还要进行一定的数据处理,如打印入口和返回等,以便检验被测模块与其下级模块的接口。

驱动模块和桩模块为程序单元的执行构成了一个完整的环境,如图 6-19 所示。驱动模块用以模拟被测单元的上层模块,测试执行时由驱动模块调用被测单元使其运行,桩模块模拟被测单元执行过程中所调用的模块,测试执行时桩模块使被测单元能完整闭合地运行。

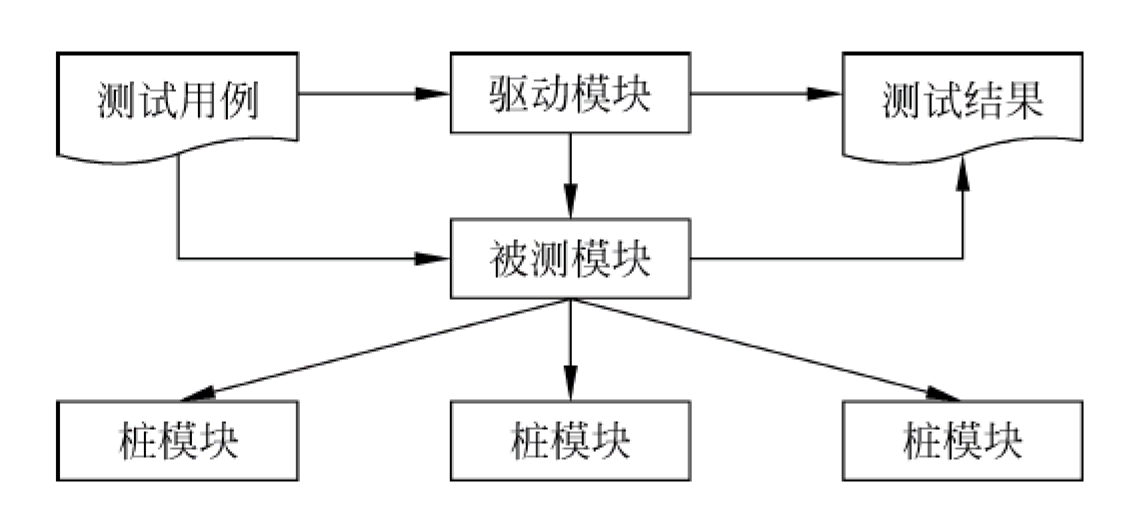


图 6-19 单元测试的测试环境

驱动模块和桩模块在软件开发结束后就不使用了,但是为了单元测试,两者都要进行开发,但是不需要与最终产品一起交付用户。因此驱动模块和桩模块的设计要尽量简单,避免因其错误而干扰被测单元的运行及测试结果判断。实际上许多程序单元不能用简单的驱动模块和桩模块进行充分的单元测试,完全的测试可以放到组装测试时再进行。

如果一个模块要完成多种功能,且以程序包或对象类的形式出现,例如 Ada 中的包、Modula 中的模块、C++中的类。这时可以将这个模块看成由几个小程序组成。对其中的每个小程序先进行单元测试要做的工作,对关键模块还要做性能测试。对支持某些标准规程的程序,更要着手进行互联测试。有人把这种情况特别称为模块测试,以区别单元测试。

2. 单元测试步骤

在单元测试中,测试用例的运行环境构建以及测试用例的设计与测试集合的准备是至关重要的。因此,单元测试可按下列步骤进行。

第一步是要构造测试用例的运行环境,即确定用例运行的前提条件,明确被测模块或被测单元所需的程序环境(全局变量赋值或初始化实体),启动测试驱动,设置桩,调用被测模块,设置预期输出条件判断,最后恢复环境(包括清除桩)。

第二步是设计"黑盒"测试用例,即接口测试用例。为此可以:设计基本功能测试用例,证明被测单元至少在某种正常情况下能够运行了;设计功能正面测试用例,找出被测单元对于设计要求的正确输入可能做出的不正确处理;设计功能反面测试用例,找出被测单元对于设计要求的错误输入可能做出的不正确处理;设计性能测试用例,找出单元对于设计要求的性能可能做不到的错误。

第三步是设计"白盒"测试用例，即覆盖测试用例，找出单元内部控制结构和数据使用可能存在的问题。

注意，在进行"白盒"测试期间，不要匆忙地删除所发现的死代码或者冗余代码，因为这很可能导致错误的产生。因为在测试别人的代码时，很可能由于测试用例不够，或者没有对被测程序整体结构的把握，而出现错误理解。

6.6 集成测试

1999 年美国火星气象卫星脱轨，用了 5 万美元进行问题查找，最后发现控制软件中有两个模块使用了不同的加速度单位。此例说明，通过单元测试的模块，还需进行集成测试。

6.6.1 集成测试的概念

集成测试又叫组装测试或联合测试，是单元测试的多级扩展，是在单元测试的基础上进行的一种有序测试。这种测试需要将所有模块按照设计要求，逐步装配成高层的功能模块，并进行测试，直到整个软件成为一个整体。集成测试的目的是检验软件单元之间的接口关系，以期望通过测试发现各软件单元接口之间存在的问题，最终把经过测试的单元组成符合设计要求的软件。集成测试验证程序和概要设计说明的一致性，任何不符合该说明的程序模块行为都应该加以记载并上报。因此，集成测试是发现和改正模块接口错误的重要阶段。

1. 为什么要开展集成测试

通常单元测试后的各个程序单元都可以正常地工作，为什么还要把它们组装在一起进行测试，看它们是否能正常工作呢？这是因为在将单元组装成一个整体时我们需要考虑相关问题：①在把各个单元模块连接起来的时候，穿越模块接口的数据是否会丢失；②一个单元模块的功能是否会对另一个单元模块的功能产生不利的影响；③各个子功能组合起来，能否达到预期要求的父功能；④全局数据结构是否有问题；⑤共享资源访问是否有问题；⑥单个模块的误差积累起来，是否会放大，从而达到不能接受的程度；⑦引入一个模块后，是否对其他与之相关的模块产生负面影响。

集成测试有以下不可替代的特点。

(1) 单元测试具有不彻底性。对于模块间接口信息内容的正确性、相互调用关系是否符合设计无能为力。只能靠集成测试来进行保障。

(2) 同系统测试相比，由于集成测试用例是从程序结构出发的，目的性、针对性更强，测试发现问题的效率较高，定位问题的效率也较高。

(3) 能够较容易地测试到系统测试用例难以模拟的特殊异常流程。从纯理论的角度来讲，集成测试能够模拟所有实际情况。

(4) 定位问题较快。由于集成测试具有可重复性强、对测试人员透明的特点，发现问题后容易定位，所以能够有效地加快进度，减少隐患。

集成测试在软件分级测试中具有很重要的意义，具体体现在：

(1) 在单元测试和系统测试间起到承上启下的作用，既能发现大量单元测试阶段不易发现的接口类错误，又可以保证在进入系统测试前及早发现错误，减少损失(事实上，对系统而言，接口错误是最常见的错误)。

(2) 单元测试通常是单人执行，而集成测试通常是多人执行或第三方执行。集成测试通过模块间的交互作用和不同人的理解和交流，更容易发现实现上、理解上的不一致和差错。

表 6-5 给出了单元测试、集成测试与系统测试的差别。

表 6-5 单元测试、集成测试与系统测试的差别

测试类型	对　　象	目　　的	测试依据	测试角度	测试方法
单元测试	模块内部的程序错误	消除局部模块的逻辑和功能上的错误和缺陷	模块逻辑设计、模块外部说明	站在开发人员角度	大量采用“白盒”测试方法
集成测试	模块间的集成和调用关系	找出与软件设计相关的程序结构、模块调用关系、模块间接口方面的问题	程序结构设计、软件概要设计说明书	站在测试人员角度	“灰盒”测试，采用较多“黑盒”方法构造测试用例
系统测试	整个系统，包括系统中的硬件等	对整个系统进行一系列的整体、有效性测试	系统结构设计、目标说明书、需求说明书等	站在用户使用角度	“黑盒”测试

因此，单元测试后，有必要进行集成测试，发现并排除在模块连接中可能发生的上述问题，最终构成要求的软件子系统或系统。

理论上凡是两个单元(如函数单元)的组合测试都可叫集成测试。但实际中的集成测试对象为模块级的集成和子系统级的集成，其中子系统集成测试称为组件测试，如图 6-20 所示。

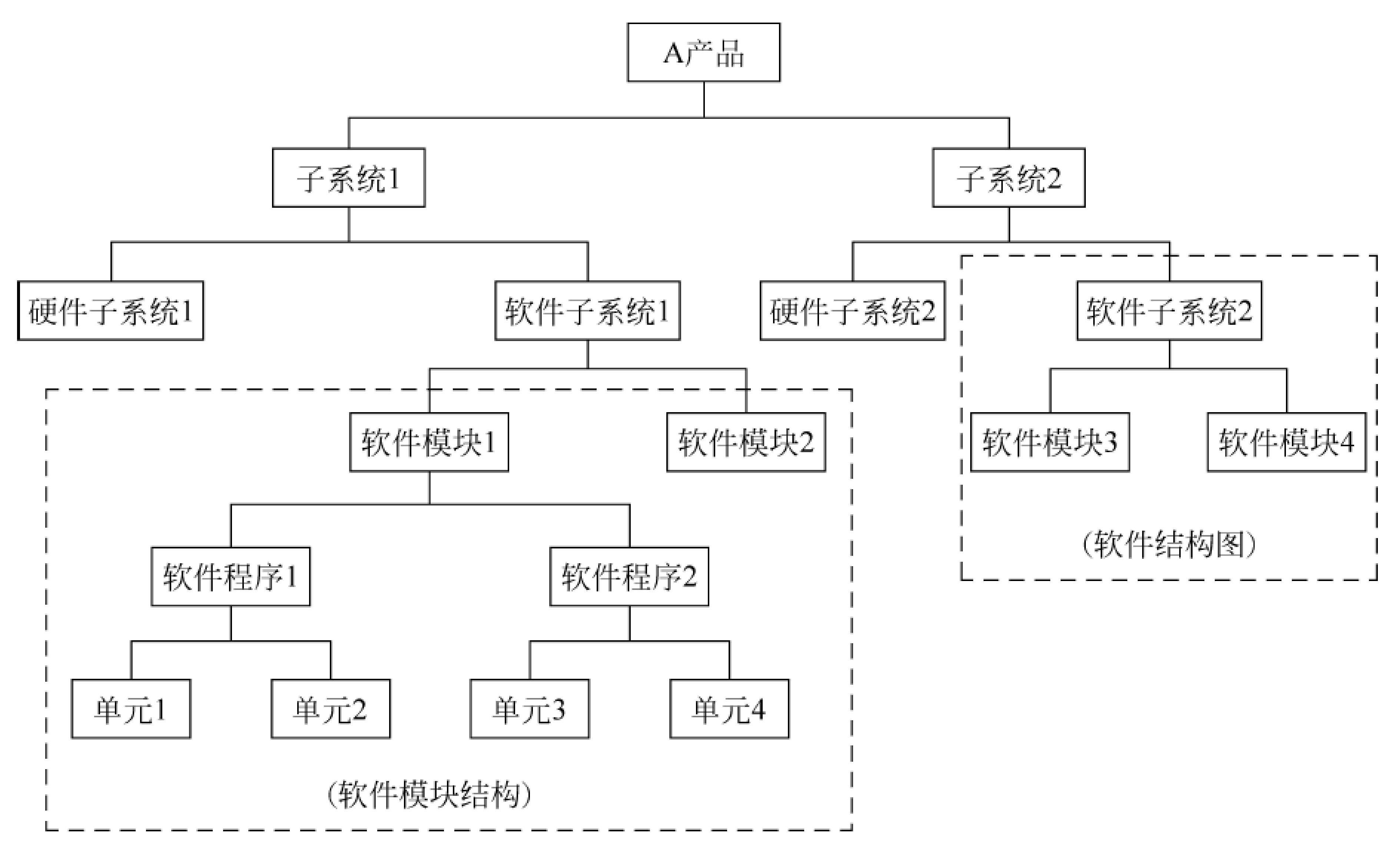


图 6-20 软件系统结构

对于传统软件来说，按集成粒度不同，可以把集成测试分为 3 个层次，即模块内集成测试、子系统内集成测试以及子系统间集成测试。

对于面向对象应用系统来说，按集成粒度不同，可以把集成测试分为两个层次：类内集成测试和类间集成测试。

2. 集成测试的内容

集成测试的内容包括以下两个方面：

(1) 功能性测试主要解决前面提到的几个问题，侧重于测试软件模块组装以后的功能有

没有达到预期效果，这是集成测试最主要的部分。

（2）集成测试还应包含其他一些测试，包括资源冲突、任务优先级冲突、性能和稳定性在内的兼容性、可靠性、可用性、效率、可维护性等测试内容。

1）集成后的功能性测试

基于集成单元实现的功能进行集成后的功能测试，考察多个模块间的协作，既要满足集成后实现的复杂功能，也不能衍生出不需要的多余功能（错误功能）。此时要关注：被测对象的各项功能是否实现；异常情况是否有相关的错误处理；模块间的协作是否高效合理。

2）接口测试

模块间的接口包括函数接口和消息接口。对函数接口的测试，应关注函数接口参数的类型和个数的一致性、输入输出属性的一致性、范围的一致性；对消息接口的测试，应关注收发双方对消息参数的定义是否一致、消息和消息队列长度是否满足设计要求、消息的完整性如何、消息的内存是否在发送过程中被非法释放、有无对消息队列阻塞进行处理等。

3）全局数据结构测试

全局数据结构往往存在被非法修改的隐患，因此对全局数据结构的测试主要关注：全局数据结构的值在两次被访问的间隔是可预知的；全局数据结构的各个数据段的内存不应被错误释放；多个全局数据结构间是否存在缓存越界；多个软件单元对全局数据结构的访问应采用锁保护机制。

4）资源测试

资源测试包括共享资源测试和资源极限测试。

共享资源测试常应用于数据库测试和支撑的测试。共享资源测试需关注：是否存在死锁现象；是否存在过度利用情况；是否存在对共享资源的破坏性操作；公共资源访问锁机制是否完善。

资源极限测试关注系统资源的极限使用情况以及软件对资源耗尽时的处理，保证软件系统在资源耗尽的情况下不会出现系统崩溃。

5）性能测试

依据某个部件的性能指标进行性能测试，及时发现性能瓶颈。

在多任务环境中，还需测试任务优先级的合理性，需考虑：实时性要求高的功能是否在高优先级任务中完成；任务优先级设计是否满足用户操作相应时间要求。

6）稳定性测试

稳定性要关注：是否存在内存泄漏而导致长期运行资源耗竭；长期运行后是否出现性能的明显下降；长期运行是否出现任务挂起等。

3. 集成测试的步骤

集成测试可按以下六步进行。

1）体系结构分析

首先，跟踪需求分析，对要实现的系统划分出结构层次图。

其次，是对系统各个组件之间的依赖关系进行分析，然后据此确定集成测试的粒度，即集成模块的大小。

2）模块分析

一般从这些角度进行模块分析：①确定本次要测试的模块；②找出与该模块相关的所有模块，并且按优先级对这些模块进行排列；③从优先级别最高的相关模块开始，把被测模块与其集成到一起；④依次集成其他模块。

3）接口分析

接口的划分要以概要设计为基础，一般通过这些步骤来完成：①确定系统的边界、子系统的边界和模块的边界；②确定模块内部的接口；③确定子系统内模块间接口；④确定子系统间接口；⑤确定系统与操作系统的接口；⑥确定系统与硬件的接口；⑦确定系统与第三方软件的接口。

4）风险分析

风险通常被分为3种类型：①项目风险（包括项目管理和项目环境的风险）；②商业风险（它和领域的相关概念及规则息息相关）；③技术风险（这是针对应用程序的具体实现而言的，主要和代码级的测试有关）。

风险分析是一个定义风险并且找出阻止潜在的问题变成现实的方法的过程。

通常把风险分析分为3个阶段：风险识别、风险评估和风险处理。

5）可测试性分析

必须尽可能早地分析接口的可测试性，提前为后续的测试工作做好准备。

6）集成测试策略分析

集成测试策略分析的主要任务就是根据被测对象选择合适的集成测试策略。

6.6.2 集成测试方法

由模块组装成程序时，有两种方法。一种方法是先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序，这种方法称为非渐增式测试方法；另一种方法是把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来进行测试，这种每次增加一个模块的方法称为渐增式测试，这种方法实际上同时完成单元测试和集成测试。

非递增式集成测试的优点是测试过程中基本不需要设计开发测试工具。不足是对于复杂系统，当出现问题时故障定位困难，和系统测试接近，难以体现和发挥集成测试的优势。

递增式集成测试逐渐集成，由小到大，边集成边测试，测完一部分，再连接一部分。在复杂系统中，划分的软件单元较多，通常是不会一次集成的。

软件集成的精细度取决于集成策略。通常的做法是先模块间的集成，再部件间的集成。这样做的优点是测试层次清晰，出现问题能够快速定位。缺点是需要开发测试驱动和桩。

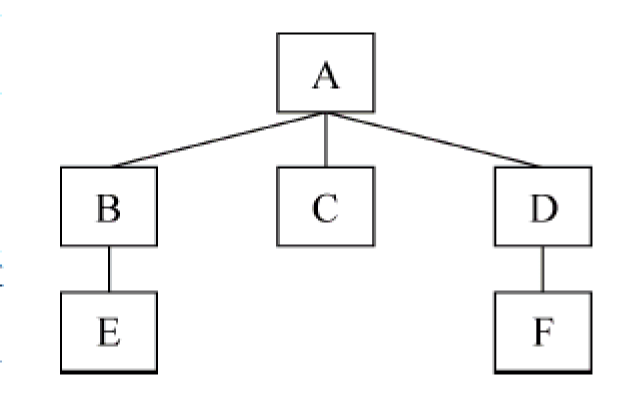


图 6-21 程序结构示意

如图6-21所示的程序结构在下面的叙述中当做示例，矩形表示程序中的单元，单元之间的连线表示程序的调用层次。图6-21中单元A调用单元B、C、D，单元B调用单元E，单元D调用单元F。

1. 一次性组装方式（或称大爆炸集成方法）

一次性组装方式是一种非渐增式测试方法，也可以叫做整体拼装，使用这种方式，首先对每个模块分别进行单元测试，对单元的测试次序是无关紧要的，它可以顺序地进行，也可以平行地进行。最后把通过单元测试的模块组装在一起进行测试，最终得到要求的软件系统。图6-22示意了这个过程。

在图6-22中，模块d1、d2、d3、d4、d5是对各个模块做单元测试所要用的驱动模块，s1、s2、s3、s4、s5是为单元测试而建立的桩模块。这种一次性组装方式试图在辅助模块的协助下，在分别完成模块单元测试的基础上，将所有测试模块连接起来进行测试。但是由于程序中不

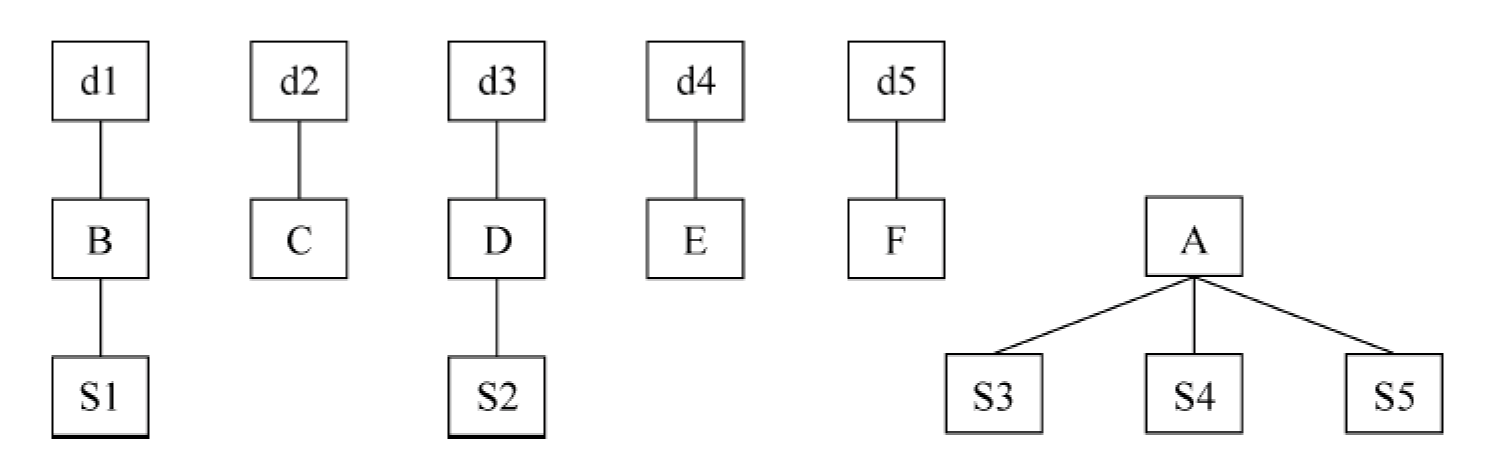


图 6-22　非渐增式集成测试

可避免地存在涉及模块间接口、全局数据结构等方面的问题,所以一次试运行成功的可能性并不是很大。其结果通常是发现错误,但是不知道去哪里找原因。查错和改错都非常困难。

非渐增式测试方法有一个贬义的名称“莽撞测试”,它的意思是一下子把几十个甚至是上百个单元很莽撞地联接在一起。显然,使用这种方法的人希望最后的组装阶段会大大缩短,并且所有的单元大体上能在一起较好地执行,这对进度比较紧的项目非常具有诱惑力,但是实际常常是相反的,用这种方法会产生一系列的问题,在我们讲述了渐增式测试方法后,通过对比就能看到这些问题。

2. 渐增式测试

渐增式测试方法不是独立地测试每个单元,而是首先把下一个要被测试的单元同已经测试过的单元集合组装起来,然后再测试,在组装的过程中边连接边测试,以发现连接过程中产生的问题,最后通过渐增式方法逐步组装成要求的软件系统。以不同的组合方式可以有很多的渐增式测试方法。典型的有自顶向下和自底向上两种。

1）自顶向下集成测试方法

自顶向下集成测试是按照程序和控制结构从主控模块开始,向下逐个把模块连接起来。把附属主控模块的子模块、孙模块等组装起来的方式有两种：深度优先和广度优先。

（1）深度优先方法。

深度优先方法(如图 6-23 所示)是先把结构中的一条主要的控制路径上的全部模块组装起来。主要路径的选择与特定的软件应用特性有关,可以尽可能地选取程序主要功能所涉及的路径。

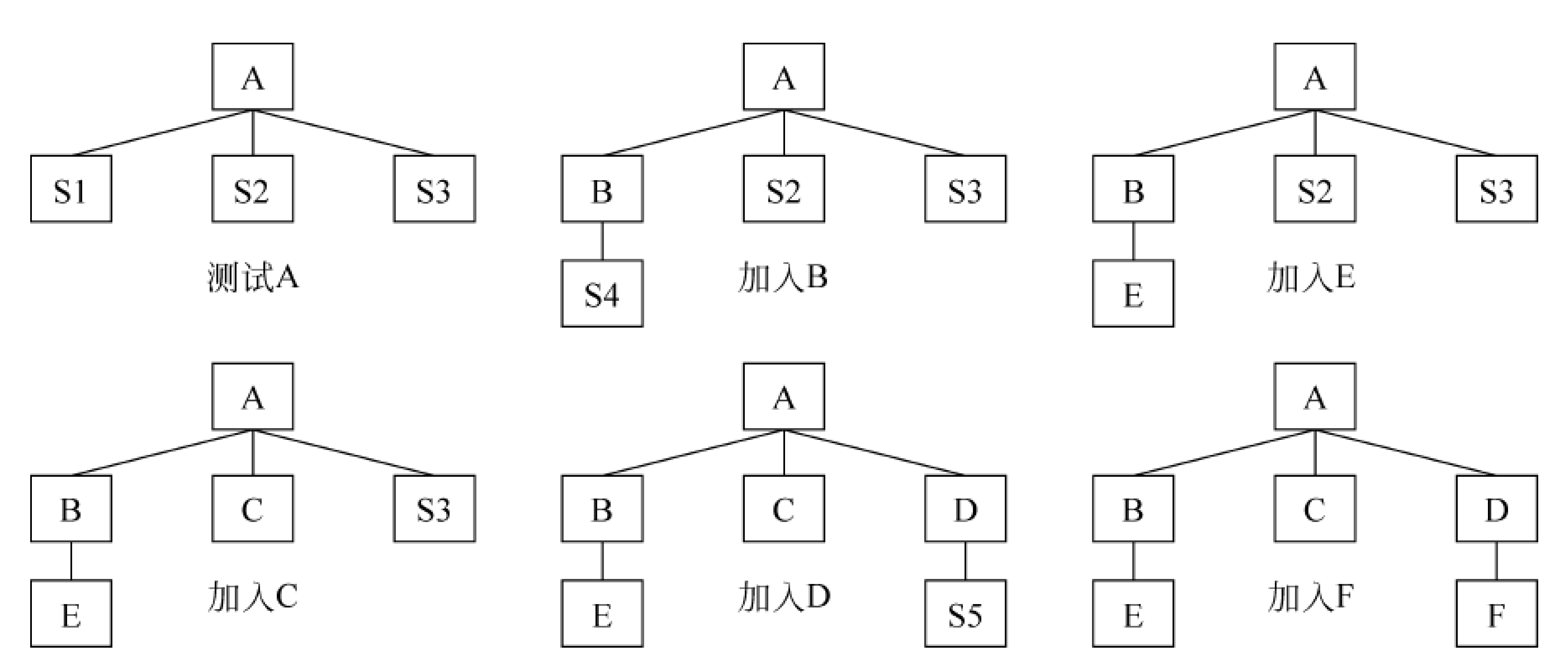


图 6-23　自顶向下按深度方向组装的例子

（2）广度优先方法。

从结构的顶层开始逐层向下组装。把上一层模块直接调用的模块组装进去,然后对每一个新组装进去的模块再把其直接调用的模块组装进去。参见图 6-24,从模块 A 出发,先组装

模块 B、C、D，接着是模块 E、F 这一层，如此类推。

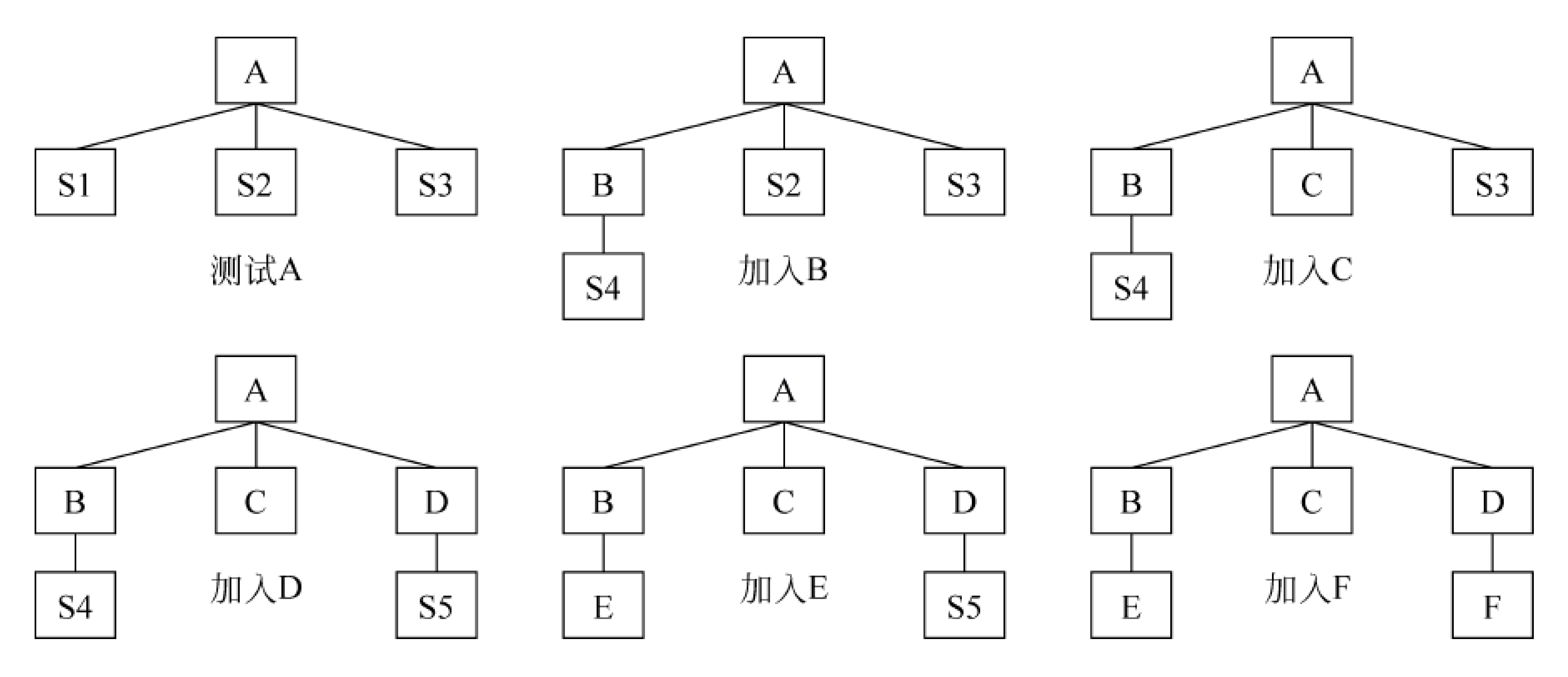


图 6-24　自顶向下按广度方向组装的例子

深度优先和广度优先自身也存在不同的组装次序，这些选择次序一般说来无所谓，但必须遵循自顶向下的原则：新组装的模块其上层必须是被测试过的。可以从以下两点来注意次序的选择。

① 如果存在程序的关键模块，我们在选择模块组装次序时，要使这些关键模块能尽早地组装进去。所谓的关键模块是指一个复杂的模块，或者包含有新的算法的模块，又或者是怀疑有错误的模块。尽早组装关键模块可以尽早地发现关键错误，在进入组装前更改单元，另外早组装进入意味着在后续的组装测试中经受更频繁的考验。

② 在设计模块组装顺序时，要尽早使 I/O 模块加入序列，形成输入→处理→输出的骨架，会使以后的测试工作简化，并减少测试的辅助性工作和人为因素所造成的测试错误和问题。

自顶向下组装测试的具体步骤为：①以主控模块作为测试驱动模块，把对主控模块进行单元测试时引入的所有桩模块用实际模块替代；②依据所选的集成策略(深度优先或广度优先)，以及新模块的选择原则，每次用一个实际单元替换一个被调用的桩模块，并开发该单元可能需要的桩模块；③每集成一个模块的同时立即进行测试，排除组装过程中可能引进的错误，如果测试发现错误，则要在修改后进行回归测试；④判断系统的组装测试是否完成，若没有完成则转到②循环进行直到集成结束。

自顶向下组装测试的优点是：①它在测试过程早期，对主要的控制点或判决点进行检验。在分解得很好的软件结构中，判决需要在结构层次的较高层确定。如果主要控制点有问题，早点认识到这个问题就变得很重要。②选用按深度方向组装的方式，可以首先实现和验证一个完整的软件功能，可先对逻辑输入的分支进行组装和测试，检查和克服隐藏的错误和缺陷，验证其功能的正确性，为此后主要分支的组装和测试提供保证。③能够较早地验证功能可行性，给开发者和用户带来成功的信心。④只有在个别情况下，才需要驱动程序(最多不超过一个)，减少了测试驱动程序开发和维护的费用。⑤可以和开发设计工作一起并行执行集成测试，能够灵活地适应目标环境。⑥容易进行故障隔离和错误定位。

自顶向下组装测试的缺点是：①在测试时需要为每个模块的下层模块提供桩模块，桩模块的开发和维护费用大；②底层组件的需求变更可能会影响到全局组件，需要修改整个系统的多个上层模块；③要求控制模块具有比较高的可测试性；④在测试较高层模块时，低层处理采用桩模块替代，不能反映真实情况，重要数据不能及时回送到上层模块，可能导致测试不充分。

解决这个问题有两种办法：①把某些测试推迟到用真实模块替代桩模块之后进行；②开发能模拟真实模块的桩模块。

自顶向下组装测试的适用范围是：①控制结构比较清晰和稳定的应用程序；②系统高层的模块接口变化的可能性比较小；③产品的低层模块接口还未定义或可能会经常因需求变更等原因被修改；④产品中的控制模块技术风险较大，需要尽可能提前验证；⑤需要尽早看到产品的系统功能行为；⑥在极限编程(extreme programming)中使用测试优先的开发方法。

2）自底向上集成测试方法

自底向上集成测试是从程序模块结构的最底层的单元开始(即这种单元不再调用其他单元)，此后选择下一个单元时就不存在唯一的方法了。其中唯一的原则是有资格作为下一个被选择的单元满足以下条件：这些单元的全部下层单元在此前已经被测试过。自底向上的测试仅需要对每个被测试模块构造一个驱动模块，而不需要构造桩模块。

自底向上集成测试的具体步骤为：

(1) 开发一个测试驱动模块，由驱动模块控制最底层模块的并行测试；也可以把最底层模块组合成实现某个子功能的模块群，由驱动模块控制它进行测试。

(2) 用真实模块代替驱动模块，与它已经通过测试的下属模块组装成为完成更大功能的新模块群。

(3) 判断程序组装的过程是否已经达到主模块，如果是则表明组装完成、测试结束，否则从(1)开始循环执行直到组装结束。

以图6-21所示的系统结构为例，用图6-25来说明自底向上集成测试的顺序。

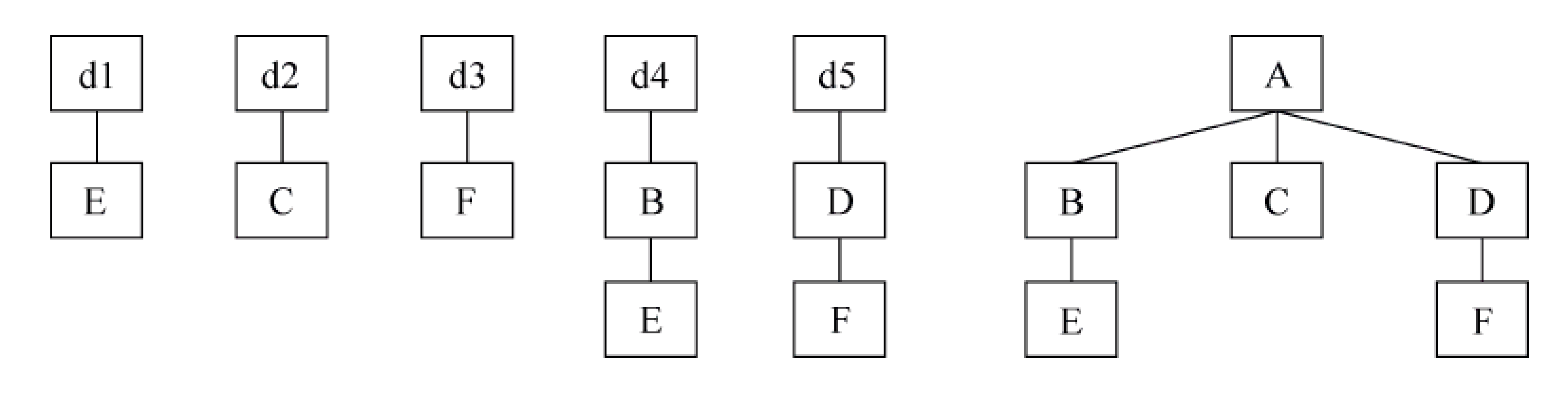


图6-25　自底向上组装或集成的例子

自底向上集成测试的优点在于：①由于驱动模块模拟了所有调用参数，测试模块返回结果不影响驱动模块，生成测试数据也没有困难；②可以尽早地验证底层模块的行为，如果关键模块是在结构图的底部，自底向上的测试是有优越性的；③自底向上的集成测试不必开发桩模块，提高了测试效率；④对实际被测模块的可测试性要求要少，容易对错误进行定位。

自底向上的集成测试的缺点是：①当最后一个模块尚未测试时，还没有呈现出被测软件系统的雏形；②只有到测试过程的后期才能发现时序问题和资源竞争问题；③驱动模块的设计工作量大；④不能及时发现高层模块设计上的错误。

因此，在测试软件系统时，应根据软件的特点和工程的进度，选用适当的测试策略，有时混合使用两种策略更为有效。

自底向上集成测试的适用范围是：①底层模块接口比较稳定的产品；②高层模块接口变更比较频繁的产品；③底层模块开发和单元测试工作完成较早的产品。

3）混合渐增式集成测试方法(或称三明治集成方法)

自顶向下集成测试方法和自底向上集成测试方法各有优缺点，一般来讲，一种方法的优点是另一种方法的缺点，因此产生了混合渐增式集成测试方法。下面介绍三种常见的混合渐增式集成测试方法。

(1) 衍变的自顶向下的渐增式测试,它的基本思想是强化对输入输出模块和引入新算法模块进行测试,再自底向上组装成为功能相当完整且相对独立的子系统,然后由主模块开始自顶向下进行渐增式测试。

(2) 自底向上—自顶向下的渐增式测试,首先对含读操作的子系统自底向上直至根节点模块进行组装和测试,然后对含写操作的子系统做自顶向下的组装与测试。

(3) 回归测试,这种方式采取自顶向下的方式测试被修改的模块及其子模块,然后将这一部分视为子系统,再自底向上测试,以检查该子系统与其上级模块的接口是否匹配。

在组装测试时,测试者应当确定关键模块,对这些关键模块及早进行测试(对该模块及其所在层下面的各层使用自底向上的集成策略,然后再对该模块所在层上面的层次使用自顶向下的集成策略,最后对系统进行整体测试)。关键模块至少应具有这几种特征:①满足某些软件需求;②在程序的模块结构中位于较高的层次(高层控制模块);③较复杂,较容易发生错误;④有明确定义的性能要求。

混合渐增式集成测试方法的优点是除了具有自顶向下和自底向上两种集成策略的优点之外,运用一定的技巧,能够减少桩模块和驱动模块的开发。其缺点则是在被集成之前,中间层不能尽早得到充分的测试。

混合渐增式集成测试方法的适用范围:多数软件开发项目都可以应用此集成测试策略。

现在我们已经分别介绍了非渐增式测试方法和渐增式测试方法,从中可以看出渐增式测试方法比非渐增式测试方法有以下优点。

(1) 非渐增式测试需要更多的工作量,对于图 6-21 所示的程序模块结构,使用非渐增式测试方法可能需要五个驱动模块和六个桩模块。但对自底向上的渐增式测试方法仅需要五个驱动模块,不需要构造桩模块,减少了辅助性测试工作。

(2) 非渐增式测试方法先分散测试,再集中起来一次完成组合和测试,如果在模块接口处存在差错,只会在最后的组合时一下子暴露出来。而使用渐增式测试方法可以较早地发现模块接口错误,这是由于较早地把模块组合起来进行测试所致。

(3) 作为一个结果,使用渐增式测试将使调试工作变得容易,因为渐增式测试逐步组合和逐步测试模块,把可能出现的错误逐步分散暴露出来,并且由于每次组合一个模块,错误发生时,可以比较容易地定位;这些错误是在最新增加的模块的连接中出现的。反之,使用非渐增式测试方法,直到对各个模块测试结束,对整个程序进行组合时才能发现错误,这时再确定错误的位置就非常困难,因为它可能出现在程序的任何地方。

(4) 渐增式测试方法利用以前已测试过的模块取代非渐增式测试方法中所需要的驱动(或桩)模块,这样对后面模块的测试会因前面模块已经实际测试过而得到更多的检验,因而使得整个程序的测试能取得较好的效果。

那么非渐增式测试方法为什么还要存在呢?一个原因是因为在实际工作中有人是这样进行程序的组装测试的,需要在这里指出其弊端;另一个原因是非渐增式测试方法在特定条件和特定范围内能起到一定作用,它把整个软件系统组装起来也很快,但是必须小心谨慎。非渐增式测试方法应用必须具备一系列的条件,但这仅仅是必要条件。那就是在一个做得很好且高度模块化的设计中,模块间的相互作用很小,而且详尽说明了接口,且将接口错误保持在最低限度,这时可以考虑用非渐增式测试方法。

6.6.3 集成测试过程

集成测试是一种正规测试过程,有着不同阶段的任务。集成测试通常划分为 4 个阶段:

集成测试计划阶段、集成测试设计与开发阶段、集成测试执行阶段、集成测试评估阶段。

集成测试通常是在开始进行体系结构设计的时候介入并制定测试方案，在进入详细设计之前完成集成测试方案，在进入系统测试之前结束集成测试。

1. 集成测试各阶段任务的划分

1）集成测试计划阶段

集成测试计划一般安排在概要设计评审通过后大约一个星期的时候，参考需求规格说明书、概要设计文档、产品开发计划时间表来制定。

集成测试计划所包含的内容有：①确定集成测试策略、方法、内容、范围、通过准则；②工具考虑，重用分析；③基于项目人力、设备、技术、市场要求等各方面决策；④集成测试进度计划；⑤工作量估算、资源需求、进度安排、风险分析和应对措施；⑥集成测试方案编制；⑦接口分析、测试项、测试特性分析（要体现测试策略）。

编制集成测试计划要与单元测试的完成时间协调起来，要考虑如下因素：①是采用何种系统组装方法来进行组装测试；②组装测试过程中要考虑集成的层次、软件的层次、模块的复杂度和重要性以及连接各个模块的顺序；③模块代码编制和测试进度是否与组装测试的顺序一致；④测试过程中是否需要专门的硬件设备。

解决了上述问题之后，就可以列出各个模块的编制、测试计划表，标明每个模块单元测试完成的日期、首次集成测试的日期、集成测试全部完成的日期，以及需要的测试用例和所期望的测试结果。

在缺少软件测试所需要的硬件设备时，应检查该硬件的交付日期是否与集成测试计划一致。例如，若测试需要数字化仪和绘图仪，则相应测试应安排在这些设备能够投入使用之时，并需要为硬件的安装和交付使用保留一段时间，以留下时间余量。此外，在测试计划中需要考虑测试所需软件（驱动模块、桩模块、测试用例生成程序等）的准备情况。

2）集成测试设计与开发阶段

一般在详细设计开始时，就可以着手进行。可以把需求规格说明书、概要设计、集成测试计划文档作为参考依据。

此时要完成测试规程/测试用例的设计与开发，确定测试步骤，设计测试数据，完成测试工具、测试驱动和桩的开发。

3）执行阶段

当所有的集成测试工作准备完毕，测试人员在单元测试完成以后就可以执行集成测试。

此时要搭建好测试环境，开展测试工作，确定测试结果，处理测试过程中的异常。

4）评估阶段

当集成测试执行结束后，要召集相关人员对测试工作进行度量，对测试结果进行评估，确定是否通过集成测试。

执行阶段的度量要采集相关数据，包括集成测试对象的数量、运行的用例数量、通过/失败的用例数量、发现的缺陷数量、遗留的缺陷数量、集成测试执行的工作量。

评估阶段要完成的工作有：按照集成测试报告模板出具集成测试报告，如有必要对集成测试报告进行评审，将所有测试相关工作产品纳入配置管理。

2. 集成测试工作开展的原则

集成测试可以在开发部进行，也可以由独立的测试部执行。开发部尽量进行集成测试，测试部可有选择地进行集成测试。

在开展集成测试时，要遵循如下的原则：

(1) 集成测试是产品研发中的重要工作,需要为其分配足够的资源和时间;
(2) 集成测试需要经过严密的计划,并严格按计划执行;
(3) 应采取增量式的分步集成方式,逐步进行软件部件的集成和测试;
(4) 应重视测试自动化技术的引入与应用,不断提高集成测试效率;
(5) 应该注意测试用例的积累和管理,方便进行回归并进行测试用例补充。

怎样才能有效地、合理地开展集成测试,即选择什么方式把模块组装起来形成一个可运行的系统,直接影响到模块测试用例的形式、所用测试工具的类型、模块编号和测试的次序、生成测试用例和调试的费用等,这些问题在集成测试开展过程中都应该考虑。

3. 集成测试工作完成的标志

怎样判定集成测试过程完成了,可按以下几个方面检查:
(1) 成功地执行了测试计划中规定的所有集成测试;
(2) 修正了所有发现的错误;
(3) 测试结果通过了专门小组的评审。

集成测试应由专门的测试小组来进行,测试小组由有经验的系统设计人员和程序员组成。整个测试活动要在评审人员出席的情况下进行。

在完成预定的组装测试工作之后,测试小组应负责对测试结果进行整理、分析,形成测试报告。测试报告中要记录实际的测试结果、在测试中发现的问题、解决这些问题的方法以及解决之后再次测试的结果。此外还应提出目前不能解决、还需要管理人员和开发人员注意的一些问题,提供测试评审和最终决策,以提出处理意见。

6.7 确认测试

集成测试实际上是按照设计的要求把所有的模块组装成一个完整的软件系统的测试,各单元之间的接口错误也已经基本排除了,接着就应该进一步验证软件的有效性,这时测试工作进入了确认阶段(或称配置项测试)。

在一些软件测试阶段划分模型中,确认测试是介于集成测试和系统测试之间必不可少的一项测试工作或任务,也是软件测试分级中的一个重要级别。

6.7.1 确认测试基本概念

确认测试是严格遵循有关标准的一种符合性测试,以确定软件产品是否满足所规定的要求。若能达到这一要求,则认为开发的软件是合格的。因而有时又将确认测试称为合格性测试。所谓规定要求指的是软件规格说明书中确定的软件功能和技术指标,或是专门为测试所规定的确认准则。

确认测试是在完成集成测试之后,依据确认测试准则,针对软件需求规格说明进行的测试,以确认所开发的软件系统能否满足规定的功能和性能需求。软件确认测试通常采用"黑盒"测试方法。测试内容主要包括安装测试、功能测试、性能测试、可靠性测试、安全性测试、效率测试、可用性测试、可移植性测试、可维护性测试、文档测试等。由于确认测试针对的是用户的直接需求,所以,应在尽可能真实的环境中进行。

确认测试必须有用户的积极参与,或者以用户为主进行。用户应该参与设计测试方案,使用用户界面输入测试数据并且分析评价测试的输出结果。为了使用户能够积极主动地参与确认测试,特别是为了使用户可以有效地使用这个软件系统,通常在验收之前由软件开发单位对

用户进行培训。

在确认测试中，α 测试是在开发场所进行的，有用户参与，而 β 测试是在客户场所进行的。

6.7.2　确认测试过程

确认测试执行的过程必须依照相关的软件测试评价标准，制定相应的测试规范，利用确认测试可能的测试方法，借助可能的测试工具，逐一测试上述标准中所有的测试项。测试的具体组织与实施是在已建立的软件测试管理体系下展开，通过确认测试确认软件是否满足用户需求，相关结论应以规范的测试报告形式给出。

在确认测试阶段需要做的工作如图 6-26 所示。首先要进行有效性测试以及软件配置复审，然后进行验收测试和安装测试，在通过了专家鉴定之后，才能成为可交付的软件。

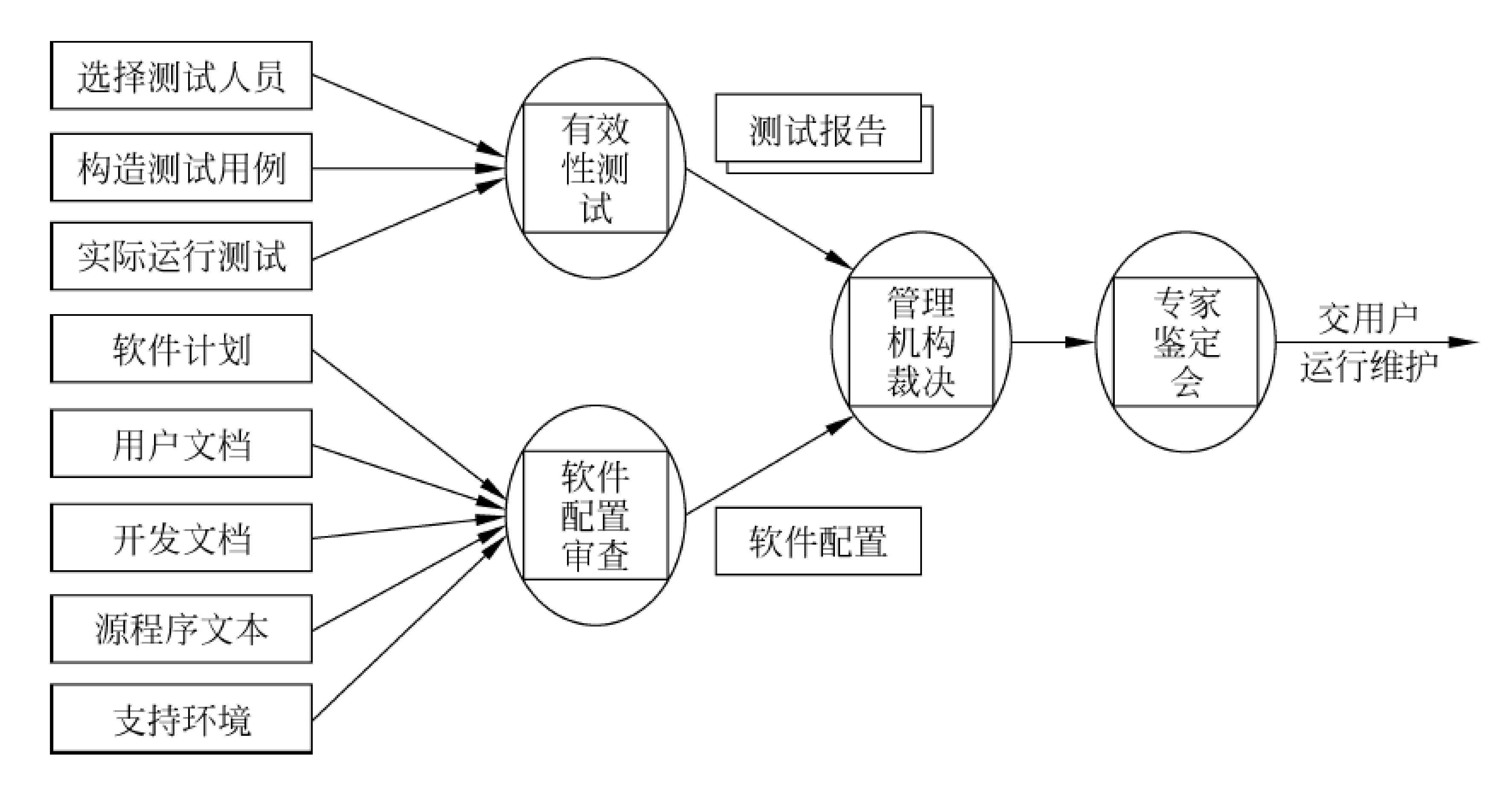


图 6-26　确认测试流程

1. 有效性测试

有效性测试又称确认测试，这是因为在软件需求规格说明中，要求描述全部用户可见的软件属性，其中有一条叫做有效性准则，它包含的信息就是软件确认测试的基础。有效性测试可在模拟的环境下，运用"黑盒"测试的方法，验证被测软件是否满足需求规格说明书列出的需求（主要任务是验证软件的功能和性能及其他特性是否与用户的要求一致）。对软件的功能和性能要求在软件需求规格说明书中已经明确规定，它包含的信息就是软件确认测试的基础。为此，在确认测试过程中，应该仔细设计测试计划和测试过程，测试计划包括要进行的测试的种类及进度安排，测试过程规定了用来检测软件是否与需求一致的测试方案。通过测试和调试要保证软件能满足所有功能要求，能达到每个性能要求，文档资料是准确而完整的，此外，还应该保证软件能满足其他预定的要求（例如，软件的安全性、可移植性、兼容性、可维护性等）。

在有效性测试中，当全部软件测试用例运行完后，会有下述两种可能的结果：

（1）功能和性能与用户要求一致，这说明软件的这部分功能或性能特征与需求规格说明书相符合，从而接受了这部分程序。

（2）功能和性能与用户要求有差距，这说明软件的这部分功能或性能特征与需求规格说明书不相符合，因此要为它提交一份问题报告。

在这个阶段发现的问题往往和需求分析阶段的差错有关，涉及的面通常比较广，因此解决起来也比较困难。为了制定解决确认测试过程中发现的软件缺陷或错误的策略，通常需要和用户进行充分的协商。

2. 软件配置复查

确认测试的另一个重要环节是配置复审。复审的目的在于保证软件配置齐全、分类有序，并且包括软件维护所必需的细节。其主要内容为文档资料，如用户所需资料(用户手册、操作手册等)、设计资料(设计说明书等)、源程序以及测试资料(测试说明书、测试报告等)。配置审查(configuration review)有时也称配置审计(configuration audit)。

除了按合同规定的内容和要求，由人工审查软件配置外，在确认测试过程中还应该严格遵循用户指南及其他操作程序，以便检验这些使用手册的完整性和正确性，且必须仔细记录发现的遗漏或错误，并适当地补充和改正。

3. α 测试和 β 测试

在软件交付使用之后，用户将如何使用程序，对于开发者来说是无法预测的。因为用户在使用过程中常常会发生对使用方法的误解、异常的数据组合，以及产生对某些用户来说似乎是清晰的但是对于某些用户来说却是难以理解的输出，等等。

如果软件是专为某个客户开发的，可以进行一系列验收测试，以便用户确认所有需求都得到满足了。验收测试是由最终用户而不是系统的开发者进行的。事实上，验收测试可以持续几个星期甚至是几个月，因此能够发现随着时间流逝可能会降低系统质量的累积错误。

但是如果一个软件是为多个用户开发的产品的话，让每个用户逐个进行正式的验收测试是不切实际的。那么在这种情况下，绝大多数软件开发商都使用被称为 α 测试和 β 测试的过程，来发现那些看起来只有最终用户才能发现的错误。

α 测试是由用户在开发环境下进行的测试，也可以是开发机构内部的用户在模拟实际操作环境下进行的测试。软件在一个自然设置状态下使用，开发者在用户旁边，对用户进行“指导”的测试，并负责记录发现的错误和使用中遇到的问题。这是在受控制的环境下进行的测试。α 测试的目的是评价软件产品的功能、可用性、可靠性、性能和支持。尤其注重产品的界面和特色。α 测试人员是除开发人员外，首先见到产品的人，他们提出的功能和修改意见是非常有价值的。

β 测试是由软件的最终用户们在一个或者多个客户场所进行。与 α 测试不同，开发者是不在 β 测试的现场，因此 β 测试是软件在开发者不能控制的环境中的“真实”应用。用户记录在 β 测试过程中遇到的一切问题，并且定期把这些问题报告给开发者。开发者在接受到 β 测试的测试报告后，对软件产品进行必要的修改，并准备向全体客户发布最终的软件产品。β 测试主要衡量产品的功能、可用性、可靠性、性能和支持。着重于产品的支持性，包括文档、客户培训和支持产品生产能力。只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。由于它处在整个测试的最后阶段，不能指望这时发现主要问题。同时，产品的所有手册文本也应该在此阶段完全定稿。

4. 验收测试

在通过了系统的有效性测试及软件配置审查之后，就应该开始系统地验收测试。验收测试是以用户为主的测试。软件开发人员和软件质量保证人员也应该参加。由用户参加设计测试用例，使用用户界面输入测试数据，并分析测试的输出结果。一般使用生产中的实际数据进行测试，在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

5. 确认测试结果

确认测试的结果分别是：①功能和性能与需求文档及用户的要求一致，软件可以接受，即通过测试；②功能和性能与需求文档及用户的要求有一定的差距，此时需要详细列出软件各项缺陷或问题的清单或列表，提交对应的问题报告。必要时，要与用户协商来解决所发现的缺

陷或错误。

确认测试应交付的文档有确认测试分析报告、最终的用户手册和操作手册、项目开发总结报告。

6.8 系统测试

由于软件只是系统中的一个组成部分，软件开发完成以后，最终还要与系统中的其他部分配套运行，进行系统测试。系统测试是软件测试分级中的一个非常重要级别。下面的示例很好地说明了系统测试的重要性。

某无人机的飞控系统在系统测试中发现软件问题50个，其中关键错误25个：①当发动机空中停止运转后，系统不能进行发动机启动；②进入失速状态后，飞机失去控制；③链路中断又恢复，飞机不接受控制，不能着陆，越飞越远。

6.8.1 系统测试概念

什么是系统测试，不同的标准其定义不一样，如《软件工程术语》GB/T 11457—1995给出的定义是：测试整个硬件和软件系统的过程，以验证系统是否满足规定的需求；《IEEE软件验证与确认标准》1998的定义是，为了验证和确认系统是否符合初始目标而对集成的软、硬件系统进行的测试活动。

目前，软件工程界对系统测试的一般性定义为：系统测试是为判断系统是否符合规定而对集成的软、硬件系统进行的测试活动。它是将已经集成好的软件系统，作为整个基于计算机系统的一个元素，与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起，在实际运行(使用)环境下，对计算机系统进行一系列的组装测试和确认测试。它是使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的系统需求或是弄清预期结果与实际结果之间的差别。

系统测试的对象是完整的、集成的计算机系统，重点是新开发的软件配置项的集合。因此，系统测试实际上是针对系统中各个组成部分进行的综合性检验。尽管每一个检验有着特定的目标，然而所有的检测工作都要验证系统中每个部分均已得到正确的实现，并能完成指定的功能。

系统测试的目的是在真实系统工作环境下通过与系统的需求定义(如功能需求)作比较，检验完整的软件配置项能否和系统正确连接，发现软件与系统/子系统设计文档和软件开发合同规定不符合或与之矛盾的地方，发现产品缺陷并度量产品质量。

另外，系统测试不仅关注系统的功能，也包括性能、安全等非功能的测试。在实际的项目里，因为时间和投资预算的关系，测试资源主要消耗在功能测试上，非功能测试经常被很多项目忽略。这确实有点令人惋惜。尽管非功能测试的确非常难做而且大多数工作需要在项目初期就开始做，但是认真对待非功能测试是非常必要的。

最后，系统测试还要检验系统的文档等各种资料是否完整、有效。

在系统测试之初，应该一方面为系统测试提供必要的软、硬件及资料支持，另一方面从软件测试角度提出系统测试中关于软件的测试设计。

系统测试的测试用例应根据需求分析说明书来设计，并在实际使用环境下来运行。通常系统测试采用“黑盒”测试技术，并由独立的测试人员完成。

从软件测试角度看，系统测试有如下两方面的意义。

(1) 系统测试的环境是软件真实运行环境的最逼真的模拟。系统测试中，各部分研制完成的真实设备逐渐替代了模拟器，是软件从未有过的运行环境。有关真实性的一类错误，包括外围设备接口、输入/输出、多处理器设备之间的接口不相容，整个系统的时序匹配等，在这种运行环境下能得到比较全面的暴露。

(2) 通常系统测试的困难在于不容易从系统目标直接生成测试用例。系统测试是由系统测试人员组织，从系统完成任务的角度开展的测试，软件在系统测试下获得了系统任务下直接的"测试实例"，这对检验软件是否满足系统任务要求是非常有意义的。

6.8.2 系统测试中关注的重要问题

按照软件测试生命周期概念，很显然，系统测试中最为关注的问题无非系统测试过程定义、系统测试需求获取、系统测试策略选择、系统测试技术与方法应用、系统测试环境建立、系统测试人员组织系统测试设计与实施以及系统测试要交付的文档等。下面，我们就这些关注的问题分别进行论述。

1. 系统测试过程定义

系统测试过程与第 8 章将要叙述的测试过程是一致的，包括制定系统测试计划、设计测试用例、实施系统测试、执行系统测试和评估系统测试五个阶段(如图 6-27 所示)。每个阶段的内容可参见表 6-6。

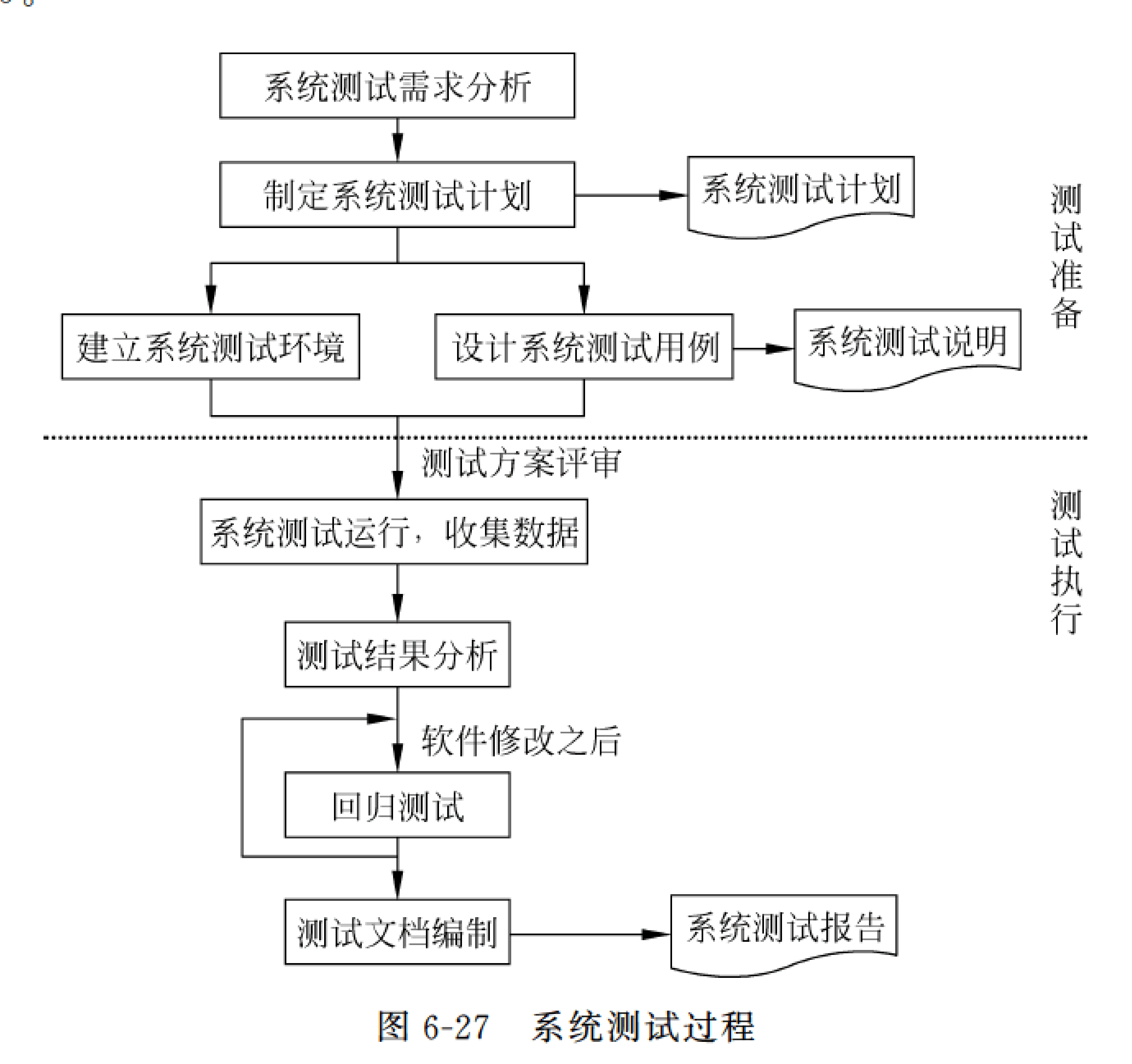


图 6-27 系统测试过程

表 6-6 系统测试各阶段的任务

活动名称	输入工作	输出工作	角色
制定系统测试计划	软件需求文档、软件项目计划	系统测试计划	测试设计员
设计系统测试	系统测试计划、软件设计文档	系统测试用例	测试设计员
实施系统测试	系统测试计划、系统测试用例	系统测试脚本	测试设计员
执行系统测试	系统测试计划、被测软件系统、系统测试用例、系统测试脚本	测试结果	测试员
评估系统测试	测试结果	软件测试报告	测试设计员、测试员

从图 6-27 可以看出，整个测试过程分为两个阶段，测试准备和测试执行。测试准备过程包括制定系统测试计划、建立系统测试环境、设计系统测试用例；测试执行过程包括测试运行、分析测试结果数据并生成软件问题清单、进行回归测试并生成测试报告。

2. 系统测试需求获取

从软件的需求规格说明和其他相关文档提取测试需求的过程是一个寻找原子系统功能的过程(一种在系统层次上可以观察得到的端口输入和输出事件的行动)，系统测试需求主要来源于需求规格说明书或系统测试项目合同等。测试需求最终体现为测试定义、测试类型、测试内容及测试对象等。在进行系统测试需求分析时，可应用以下几条规则：

(1) 测试需求必须是可观测、可测评的行为。对于不能观测或测评的测试需求，是无法对其评估的，因此也就无法确定需求是否已经满足。

(2) 在每个用例或系统的补充需求与测试需求之间不存在一对一的关系。一个用例通常具有多个测试需求；有些补充需求将派生一个或多个测试需求，而其他补充需求(如市场需求或包装需求)将不派生任何测试需求。

(3) 在需求规格说明书中每一个功能描述将派生一个或多个测试需求，性能描述、安全性描述等也将派生出一个或多个测试需求。

(4) 在系统测试需求中，以传统测试类型中的功能性测试需求和性能测试需求最为重要，是整个系统测试需求中的核心。

1) 功能性测试需求

功能性测试需求来自测试对象的功能性说明。每个用例至少会派生一个测试需求。对于每个用例事件流，测试需求的详细列表至少会包括一个测试需求。对于需求规格说明书中的功能描述，将至少派生一个测试需求。

2) 性能测试需求

性能测试需求来自测试对象的指定性能行为。性能通常被描述为响应时间和资源使用率的某种评测。性能需要在各种条件下进行评测，这些条件包括：①不同的工作量和/或系统条件；②不同的用例和功能不同的配置。

性能需求在补充规格或需求规格说明书中的性能描述部分中说明。对包括这些内容的语句要特别注意：①时间语句，如响应时间或定时情况；②指出在规定时间内必须出现的事件数或用例数的语句；③将某一项性能的行为与另一项性能的行为进行比较的语句；④将某一配置下的应用程序行为与另一配置下的应用程序行为进行比较的语句；⑤一段时间内的操作可靠性(平均故障时间 MTBF 或平均无故障时间 MTTF)；⑥配置或约束。

3) 其他测试需求

其他测试需求包括配置测试、安全性测试、容量测试、强度测试、故障恢复测试、负载测试等，这些测试需求可以从非功能性需求中发现与其对应的描述。每一个描述信息可以生成至少一个测试需求。

3. 系统测试策略选择

测试策略用于说明某项特定测试工作的一般方法和目标。系统测试策略主要针对系统测试需求确定测试类型及实施测试的方法和技术。一个好的测试策略应该包括：要实施的测试类型和测试目标，采用的技术，用于评估测试结果和测试是否完成的标准，以及对测试策略所述的测试工作存在影响的特殊事项。

确定系统测试策略首先应清楚地说明所实施系统测试的类型和测试的目标。清楚地说明这些信息有助于尽量避免混淆和误解(尤其是由于有些测试类型看起来非常类似，如强度测试

和容量测试)。测试目标应该表明执行测试的原因。

系统测试的测试类型一般包括功能测试、性能测试、负载测试、强度测试、容量测试、安全性测试、配置测试、故障恢复测试、安装测试、文档测试、用户界面测试等,其中,功能测试、配置测试、安装测试等在一般情况下是必需的,而其他的测试类型则需要根据软件项目的具体要求进行裁剪。

4. 系统测试采用的技术

系统测试主要采用"黑盒"测试技术设计测试用例来确认软件满足需求规格说明的要求。

5. 系统测试环境建立

被测软件的可能运行的环境分别是开发环境、测试环境、用户环境。开发环境往往与用户环境有所差别。一个规划良好的测试环境要接近用户环境,但也要兼顾开发环境。测试环境在测试计划和测试用例中要事先定义和规划。

建立系统测试环境要考虑下列因素:

(1) 确定硬件环境和软件环境。这里,硬件环境指测试必需的服务器、客户端、网络连接设备,以及打印机/扫描仪等辅助硬件设备所构成的环境,软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。

(2) 规划系统测试环境。分析用户环境中哪些配置可能对软件有所影响,并在此基础上规划和建立测试环境。

(3) 建立测试环境需要考虑的因素。建立测试环境需要考虑计算机平台、操作系统、浏览器、软件支持平台、外围设备、网络环境、数据环境、其他专用环境等因素。

(4) 确定建立系统测试环境的步骤。如安装应用程序、安装和开发测试工具、设置专用文件,包括将这些文件与测试所需的数据相对应、建立与应用程序通信的实用程序、配备适当的硬件以及必要的设备等。

6. 系统测试人员组织

系统测试至少需要由一个独立的测试组来开展工作,或者由项目组为每一个软件项目成立测试组,确定测试经理(通常由测试设计员担任)一名,测试设计员和测试员多名。

测试团队一般 4～5 人,否则应该细分为测试组。测试经理/测试组长制定测试计划和测试方案,分配测试任务并检查测试进度,代表测试团队与开发团队、产品销售团队、用户沟通,开展实际测试,最后评估系统测试;测试设计员参与系统测试计划和方案的制定,按照系统测试方案设计测试用例和测试代码、设计所需测试工具、编写测试规程、完成系统测试报告并组织评审、输出测试用例和总结等经验文档;测试员执行系统测试用例、填写缺陷报告并检查缺陷处理结果。图 6-28 是系统测试过程中各阶段测试人员所担当的工作。

根据相关统计数据我们知道,测试人员的效率是平均每个工作日发现 3～5 个错误;开发人员平均每修正 3 个错误,会引进 1 个新的错误;平均 75%的错误会在单元测试阶段解决;平均 20%的错误会在集成测试和系统测试阶段解决;平均 5%的 Bug 会被交付给用户——普通大型民用软件平均错误率 5 个/10 000LOC,电信/银行/操作系统等软件平均错误率 5 个/100 000LOC。

软件测试与软件开发人员的配备与产品大小、复杂度、质量要求相关。目前国内外软件测试与软件开发人员的比例相差很远。在软件产业发达国家,软件企业一般是把 40%的工作花在测试上,测试人员和开发人员之比平均在 1∶1 以上。

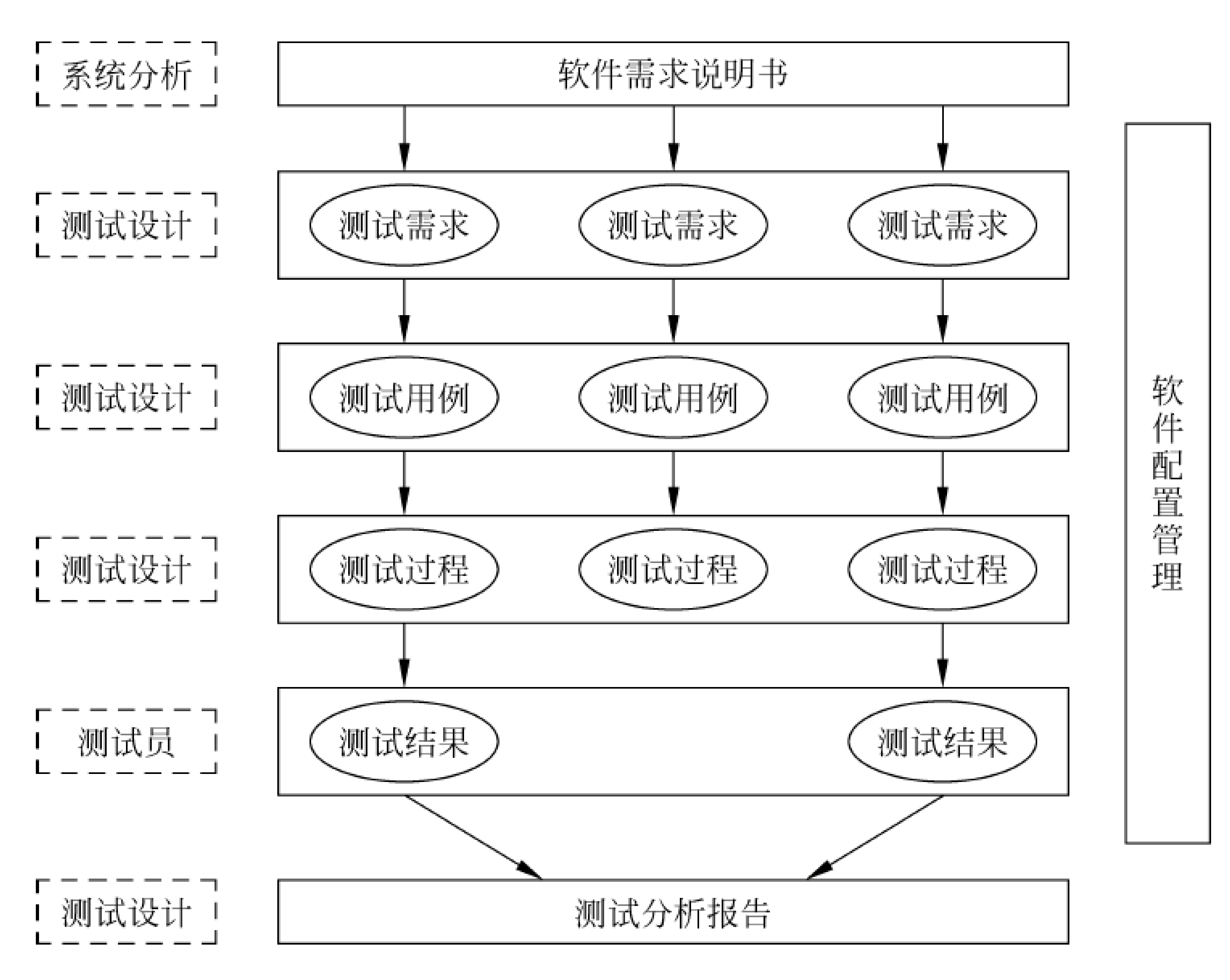


图 6-28　系统测试过程中各阶段测试人员要做的工作

7. 系统测试要交付的文档

系统测试要交付的文档主要有系统测试计划、系统测试计划评审报告、系统测试用例、系统测试用例评审报告、系统测试脚本、系统测试脚本评审报告、系统测试报告、系统测试报告评审报告、缺陷问题单若干等。

上诉报告可以根据需要进行裁减或改造，如交付系统测试计划、系统测试设计、系统测试结果、缺陷问题等报告。

6.8.3　系统测试的要求和主要内容

现代系统测试要求依据软件质量特性/子特性来进行，重点是新开发的软件配置项的集合。但在实际测试中是针对传统测试中的各种测试类型(软件质量特性/子特性与软件测试类型的对应关系可参见表 4-1)，如功能测试、可靠性测试、性能测试、安全性测试、边界测试、余量测试、恢复性测试、接口测试和强度测试等。

1. 系统测试一般要求

系统测试一般应符合以下技术要求：

(1) 系统的每个特性应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖。

(2) 测试用例的输入应至少包括有效等价类值、无效等价类值和边界数据值。

(3) 应逐项测试系统/子系统设计说明规定的系统的功能、性能等特性。

(4) 应测试软件配置项之间及软件配置项与硬件之间的接口。

(5) 应测试系统的输出及其格式。

(6) 应测试运行条件在边界状态和异常状态下，或在特定的状态下，系统的功能和性能。

(7) 应测试系统访问和数据安全性。

(8) 应测试系统的全部存储量、输入输出通道和处理时间的余量。

(9) 应按系统或子系统设计文档的要求，对系统的功能、性能进行强度测试。

(10) 应测试设计中用于提高系统安全性、可靠性的结构、算法、容错、冗余、中断处理等方案。

(11) 对完整性级别高的系统，应对其进行安全性、可靠性分析，明确每一个危险状态和导致危险的可能原因，并对此进行针对性的测试。

(12) 对有恢复或重置功能需求的系统，应测试其恢复或重置功能和平均恢复时间，并且对每一类导致恢复或重置的情况进行测试。

(13) 对不同的实际问题应外加相应的专门测试。

对具体的系统，可根据软件测试合同(或项目计划)及系统的重要性、完整性级别等要求对上述内容进行剪裁或修订。

2. 国标 GB/T 16620 中的系统测试内容

依据国标 GB/T 16620 针对系统测试的测试内容，系统测试主要从适应性、准确性、互操作性、安全保密性、成熟性、容错性、易恢复性、易理解性、易学性、易操作性、吸引性、时间特性、资源利用性、易分析性、易改变性、稳定性、易测试性、适应性、易安装性、共存性、替换性和依从性等方面(有选择的)来考虑。

对具体的系统，可根据测试合同(或项目计划)及系统/子系统设计文档的要求对上述测试内容进行剪裁。

1) 功能性

(1) 从适应性方面考虑，应测试系统/子系统设计文档规定的每一项功能。

(2) 从准确性方面考虑，可对系统中具有准确性要求的功能和精度要求的项(如数据处理精度、时间控制精度、时间测量精度)进行测试。

(3) 从互操作性方面考虑，可测试系统/子系统设计文档、接口需求规格说明文档和接口设计文档规定的系统与外部设备的接口、与其他系统的接口。包括：①测试其格式和内容，包括数据交换的数据格式和内容；②测试接口之间的协调性；③测试软件对系统每一个真实接口的正确性；④测试软件系统从接口接收和发送数据的能力；⑤测试数据的约定、协议的一致性；⑥测试软件系统对外围设备接口特性的适应性。

(4) 从安全保密性方面，可测试系统及其数据访问的可控制性。包括：①测试系统防止非法操作的模式，包括防止非授权的创建、删除或修改程序或信息，必要时做强化异常操作的测试；②测试系统防止数据被讹误和被破坏的能力；③测试系统的加密和解密功能。

2) 可靠性

(1) 在成熟性方面，可基于系统运行剖面设计测试用例，根据实际使用的概率分布随机选择输入，运行系统。其测试要求有：①测试系统满足需求的程度并获取失效数据，其中包括对重要输入变量值的覆盖、对相关输入变量可能组合的覆盖、对设计输入空间与实际输入空间之间区域的覆盖、对各种使用功能的覆盖、对使用环境的覆盖；②应在有代表性的使用环境中以及可能影响系统运行方式的环境中运行软件，验证系统的可靠性需求是否正确实现；③对一些特殊的系统，如容错软件、实时嵌入式软件等，由于在一般的使用环境下常常很难在软件中植入差错或故障，应考虑多种测试环境；④测试系统的平均无故障时间；⑤选择可靠性增长模型，通过检测到的失效数和故障数，对系统的可靠性进行预测。

(2) 从容错性方面考虑，可测试：①系统对中断发生的反应，系统在边界条件下的反应，系统的功能、性能的降级情况，系统的各种误操作模式，系统的各种故障模式(如数据超范围、死锁)；②在多机系统出现故障需要切换时系统的功能和性能的连续平稳性(注：可用故障树分析技术检测误操作模式和故障模式)。

(3) 从易恢复性方面考虑,可测试:具有自动修复功能的系统的自动修复的时间,系统在特定的时间范围内的平均宕机时间,系统在特定的时间范围内的平均恢复时间,系统的重新启动并继续提供服务的能力,系统的还原功能的还原能力。

3) 可用性

(1) 从易理解方面考虑,检查系统的各项功能,确认它们是否容易被识别和被理解。包括:①对要求具有演示功能的能力,确认演示是否容易被访问、演示是否充分和有效;②界面的输入和输出,确认输入和输出的格式和含义是否容易被理解。

(2) 从易学性方面考虑,可测试:①系统的在线帮助,确认在线帮助是否容易定位,是否有效;②还可以对照用户手册或操作手册执行系统,测试用户文档的有效性。

(3) 从易操作性方面考虑,输入数据,确认系统是否对输入数据进行有效性检查。包括:①要求具有中断执行的功能,确认它们能否在动作完成之前被取消;②要求具有还原能力(数据库的事务回滚能力)的功能,确认它们能否在动作完成之后被撤销;③包含参数设置的功能,确认参数是否已选择、是否有默认值;④要求具有解释的消息,确认它们是否明确;⑤要求具有界面提示能力的界面元素,确认它们是否有效;⑥要求具有容错能力的功能和操作,确认系统能否提示出错的风险、能否容易纠正错误的输入、能否从差错中恢复;⑦要求具有定制能力的功能和操作,确认定制能力的有效性;⑧要求具有运行状态监控能力的功能,确认它们的有效性(注:以正确操作、误操作模式、非常规模式和快速操作为框架设计测试用例,误操作模式有错误的数据类型作参数、错误的输入数据序列、错误的操作序列等。如有用户手册或操作手册,可对照手册逐条进行测试)。

(4) 从吸引性方面考虑,可测试系统的人机交互界面能否定制。

4) 效率

(1) 从时间特性方面考虑,可测试:①系统的响应时间、平均响应时间、响应极限时间;②系统的吞吐量、平均吞吐量、极限吞吐量;③系统的周转时间、平均周转时间、周转时间极限。

注:响应时间指系统为完成一项规定任务所需的时间;平均响应时间指系统执行若干并行任务所需的平均时间;响应极限时间指在最大负载条件下,系统完成某项任务需要时间的极限;吞吐量指在给定的时间周期内系统能成功完成的任务数量;平均吞吐量指在一个单位时间内系统能处理并发任务的平均数;极限吞吐量指在最大负载条件下,在给定的时间周期内,系统能处理的最多并发任务数;周转时间指从发出一条指令开始到一组相关的任务完成的时间;平均周转时间指在一个特定的负载条件下,对一些并发任务,从发出请求到任务完成所需要的平均时间;周转时间极限指在最大负载条件下,系统完成一线任务所需要时间的极限。

在测试时,应标识和定义适合于软件应用的任务,并对多项任务进行测试,而不是仅测一项任务。

注:软件应用任务的例子,如在通信应用中的切换、数据包发送、在控制应用中的事件控制,在公共用户应用中由用户调用的功能产生的一个数据的输出等。

(2) 从资源利用性方面考虑,可测试系统的输入输出设备、内存和传输资源的利用情况,包括:①执行大量的并发任务,测试输入输出设备的利用时间;②在使输入输出负载达到最大的系统条件下,运行系统,测试输入输出负载极限;③并发执行大量的任务,测试用户等待输入输出设备操作完成需要的时间(注:建议调查几次测试与运行实例中的最大时间与时间分布);④在规定的负载下和在规定的时间范围内运行系统,测试内存的利用情况;⑤在最大负载下运行系统,测试内存的利用情况;⑥并发执行规定的数个任务,测试系统的传输能力;⑦在系统负载最大的条件下和在规定的时间周期内,测试传输资源的利用情况;⑧在系统传

输负载最大条件下,测试不同介质同步完成其任务的时间周期。

5）维护性

(1）从易分析性方面考虑,可设计各种情况的测试用例运行系统,并监测系统运行状态数据,检查这些数据是否容易获得、内容是否充分。如果软件具有诊断功能,应测试该功能。

(2）从易改变性方面考虑,可测试能否通过参数来改变系统。

(3）从易测试性方面考虑,可测试软件内置的测试功能,确认它们是否完整和有效。

6）可移植性

(1）从适应性方面考虑,可测试：①软件对诸如数据文件、数据块或数据库等数据结构的适应能力；②软件对硬件设备和网络设施等硬件环境的适应能力；③软件对系统软件或并行的应用软件等软件环境的适应能力；④软件是否已移植。

(2）从易安装性方面考虑,可测试软件安装的工作量、安装的可定制性、安装设计的完备性、安装操作的简易性、是否容易重新安装(安装设计的完备性可分为三级：①最好——设计了安装程序,并编写了安装指南文档；②好——仅编写了安装指南文档；③差——无安装程序和安装指南文档。安装操作的简易性可分为四级：①非常容易——只需启动安装功能并观察安装过程；②容易——只需回答安装功能中提出的问题；③不容易——需要从表或填充框中看参数；④复杂——需要从文件中寻找参数,改变或写它们)。

(3）从共存性方面考虑,可测试软件与其他软件共同运行的情况。

(4）当替换整个不同的软件系统和用同一软件系列的高版本替换低版本时,在易替换性方面,可考虑测试：①软件能否继续使用被其替代的软件使用过的数据；②软件是否具有被其替代的软件中的类似功能。

3. 不同测试类型的测试要求

在系统测试中,对于具体的测试类型,其测试内容及测试要求如下。

1）功能测试

目标：对产品的功能进行测试,检验是否实现、是否正确实现；

方法：覆盖产品的功能；

工具：回归测试时可以使用工具。

2）性能测试

目标：对产品的性能进行测试,检验是否达标、是否能够保持；

方法：覆盖系统的性能需求,一般和负载测试结合使用；

工具：在需要大访问量时尤其需要使用工具。

3）负载测试

目标：在人为设置的高负载(大数据量、大访问量)的情况下,检查系统是否发生功能或者性能上的问题；

方法：人为生成大数据量,并利用工具模拟频繁并发访问；

工具：一般需要使用工具。

4）压力测试

目标：在人为设置的系统资源紧缺情况下,检查系统是否发生功能或者性能上的问题；

方法：人为减少可用的系统资源,包括内存、硬盘、网络、CPU 占用、数据库反应时间等；

工具：一般需要使用工具。

5）疲劳测试

目标：在一段时间内(经验上一般是连续 72 小时)保持系统功能的频繁使用,检查系统是

否发生功能或者性能上的问题；

方法：人为设置不同功能的连续重复操作；

工具：一般需要使用工具。

6）可用性测试

目标：检查系统界面和功能是否容易学习、使用方式是否规范一致，是否会误导用户或者使用模糊的信息；

方法：可以采用用户操作、观察（录像）、反馈并评估的方式，一般与功能测试结合使用。

7）安装测试

目标：检查系统安装是否能够安装所有需要的文件/数据并进行必要的系统设置，检查系统安装是否会破坏其他文件或配置，检查系统安装是否可以中止并恢复现场，检查系统是否能够正确卸载并恢复现场，检查安装和卸载过程的用户提示和功能是否出现错误。有时候将安装测试作为功能测试的一部分。

8）配置测试

目标：在不同的硬件配置下，在不同的操作系统和应用软件环境中，检查系统是否发生功能或者性能上的问题；

方法：一般需要建立测试实验室。

9）文档测试

目标：检查系统的文档是否齐全，检查是否有多余文档或者死文档，检查文档内容是否正确/规范/一致等；

方法：一般由单独的一组测试人员实施。

10）安全测试（包括病毒、加密、权限）

目标：检查系统是否有病毒，检查系统是否正确加密，检查系统在非授权的内部或外部用户访问或故意破坏时是否出现错误。

11）恢复测试

目标：在人为发生系统灾难（系统崩溃、硬件损坏、病毒入侵等）的情况下，检查系统是否能恢复被破坏的环境和数据。

12）回归测试

定义回归测试是一种选择性重新测试，目的是检测系统或系统组成部分在修改期间产生的缺陷，用于验证已进行的修改并未引起不希望的有害效果，或确认修改后的系统或系统组成部分仍满足规定的要求。

目标：检查系统变更之后是否引入新的错误或者旧的错误重新出现，尤其是在每次生成系统之后和稳定期测试的时候；

工具：一般使用工具，一般依赖于测试用例库和缺陷报告库。

13）健全测试

目标：检查系统的功能和性能是否基本可以正常使用，来确定是否可以继续进行系统测试的其他内容；

方法：正常安装，并使用正常情况下的测试用例对主要功能进行测试；同时检查系统文档是否齐全。

14）交付测试

目标：关闭所有缺陷报告，确保系统达到预期的交付标准。

方法：一般需要结合回归测试，并谨慎处理新出现的 Bug。交付测试也称为稳定期测试，

有时候与系统测试独立划分。

15）演练测试

目标：在交付给用户之前，利用相似的用户环境进行测试。例如：奥运会 MIS 系统在 2008 年前用于其他比赛。

16）背靠背测试

目标：设置一组以上的测试团队，在互相不进行沟通的情况下独立进行相同的测试项目，用来评估测试团队的效果并发现更多的错误。开始用于测试外包，现在也用于内部测试。

17）度量测试

目标：在系统中人为地放入错误（播种），并根据被发现的比例来确定系统中遗留的错误数量。开始用于测试外包，现在也用于内部测试。

18）比较测试

目标：与竞争产品以及本产品的旧版本测试同样的内容，来确定系统的优势和劣势。严格地说，比较测试属于系统测评的内容，BenchMarking 是一种特殊的比较测试。

上述 18 种测试内容并不是都要进行的，制定测试策略和测试计划的时候要有不同的侧重点，而这与测试目标、测试资源、软件系统特点和业务环境有关。

另外，上述 18 种测试最好由独立的第三方进行测试。因为进行独立测试的目的是进一步加强软件质量保证工作，提高软件的质量，并对软件产品进行客观评价。而进行第三方独立测试通常有发挥专业技术优势和独立性优势，以及促进承办方的工作等方面的优势。

6.8.4 系统测试设计

在系统测试需求确定后，可以开始测试设计工作了。而测试设计又是整个测试过程中非常重要的一个环节，测试设计的输出结果是测试执行活动依赖的执行标准，测试设计的充分性决定了整个系统测试过程的测试质量。因此，为了保证系统测试质量，必须在测试设计阶段就对系统进行严密的测试设计。

测试设计一般的流程是：首先理解软件和测试目标，然后设计测试用例，接着运行测试用例并处理测试结果，最后评估测试用例和测试设计。

在确定测试设计流程时既强调目的性也强调计划性，目标是追求测试的高效率和测试的理想结果。当然，水能载舟，亦能覆舟。文档化和按部就班方式可以降低管理难度，增强计划性，但也可能影响测试人员的经验作用和灵感。

1. 理解软件和测试目标

目的：建立软件故障模型，了解测试目标，确定测试策略和测试计划。

任务：①了解软件的功能和业务背景、用户环境；②了解软件的开发背景和系统结构、技术选型；③了解软件的质量历史、版本变化；④了解系统的测试目标和资源限制，确定测试策略和测试计划。

方法：①阅读软件使用手册，理解软件运行环境和用户行为，了解同类软件的功能和使用，了解软件要解决的问题域和解域（业务背景知识）；②试运行软件，从中熟悉软件功能，确定软件基本可以测试；③了解软件体系结构、技术选型、开发环境和工具；④阅读早期版本测试报告，以及单元和集成测试报告；⑤确定测试人员限制和时间限制，制定初步测试策略和测试计划，确定测试结束标准。

结果：①建立错误模型，指导回答该软件可能的错误会出现在哪里（用户环境与测试环境不一致会不会出问题，没有测试过的代码或功能里面会不会出问题，没有测试过的输入组合、

极端环境或功能使用方法、使用顺序中会不会出问题)，我们如何做才能发现这些错误等问题；②在了解测试目标和资源限制之后，按照错误的性价比制定设计测试用例的优先级，并确定初步的测试策略和测试计划；③确定测试结束标准或测试退出机制(已经解决的错误没有重现，所有缺陷报告已经关闭，所有测试用例全部执行完毕，通过错误播种、错误发生曲线分析、历史数据等相应方法统计出所遗留的未发现错误数量可以被接受，以及不属于技术层面且实际表明测试失败或者部分失败的市场和管理因素、预算和时间用完等因素)。

2. 设计测试用例

目的：设计尽可能多、快、好、省发现错误的测试用例(即能够找到尽可能多的、以至于所有的 BUG；能够尽可能快或早地发现最严重的 BUG；找到的 BUG 是关键的、用户最关心，且找到 BUG 后能够重现其 BUG，并为修正 BUG 提供尽可能多的信息；能够用最少的时间、人力和资源发现 BUG，且测试的过程和数据可以重用)。

任务：理解故障模型，理解现有的测试用例库，设计具体的测试用例。

方法：采用基于故障模型如经验、历史数据/错误、软件开发和运行环境的软件攻击法(这需要创造性思维，而且要注意保证满足测试用例多、快、好、省的要求，并要有以孙子兵法进行指导的战役思想，当然还要记住没有“银弹”的教诲)。

结果：测试用例(IE 4.0 的测试用例数目：10 万)。

测试用例设计的过程是系统测试过程中一个非常重要的环节，它要求从测试的角度对测试计划中的测试需求进行功能和各种特性的细化，确定出与被测功能相关的输入输出变量。

继而判断这些变量如何从测试环境中通过硬件接口输入到被测软件中，以及如何从被测软件的输出中得到。在测试说明中需要最终确定本次测试要测试的系统功能、每一个功能涉及的输入输出变量以及这些变量取值的等价类划分等。

测试用例没有标准文档格式，对于特殊人或在特殊情况下可以在运行后再形成文档。

测试用例文档由简介和测试用例两部分组成：简介部分描述了测试目的、测试范围、定义术语、参考文档、概述等；测试用例部分逐一列示各测试用例(包含的要素：标题和编号、版本号、修改记录等，针对目标和假设前提/可能发现的错误，输入和数据/代码，测试步骤，预期输出和错误发现方法)。表 6-7 是一个简单的测试用例设计表格。

表 6-7　测试用例设计表

测试用例 ID	输　入			预期结果			实际结果			测试统计		
	利率/%	贷款期限/年	贷款金额/元	月支付	总支付	总利息	月支付	总支付	总利息	通过/失败	测试日期	测试人员
TC-001	8	30	80 000	578.01								
TC-002	8.5	30	80 000	615.13								
TC-003	8.5	15	80 000	787.79								

3. 运行测试用例并处理测试结果

目的：使用测试用例发现错误并关闭错误。

任务：①运行测试用例并记录结果；②评估测试结果并记录缺陷；③处理缺陷直至缺陷关闭(即修改、延迟处理、不修改、不是错误)。

方法：①选择测试用例库中的测试用例运行；②选择新设计的测试用例运行；③录制/回放或笔录中间步骤和结果，记录下执行过程中的灵感(但不要轻易修改本次执行任务)；④分析测试结果并尽量重现和优化错误步骤，详细填写缺陷报告并提供尽可能多的信息(如尽可能

提供错误分析和修改建议），认真审核错误处理结果并及时关闭缺陷报告。

结果：①记录下的运行结果；②记录下的新的测试用例设计思路；③提交并处理的缺陷报告。

4. 评估测试用例和测试设计

目的：检验测试用例和测试设计中测试策略的有效性，必要时对测试用例和测试策略进行完善和修改，增加测试经验。

任务：①根据测试结果完善、修改、合并测试用例，如果没有文档化测试用例，此时需要文档化；②对测试用例库进行维护，即增加新的测试用例（尤其是已经发现了错误的测试用例），删除不必要的测试用例（要谨慎，除非是功能改变），修改刚刚使用的测试用例（根据测试结果），合并部分测试用例；③根据测试结果完善和修正测试策略和测试计划、产生新的测试用例设计思路。

方法：基于经验（发现了什么问题，这种问题出现的原因是什么，为什么测试用例会发现这种问题，还可以更快地发现吗，测试用例可以合并吗，可能联想到还会出现什么问题——新的测试用例），流程控制/尤其是测试用例库的维护可以借助于工具实现。

结果：①优化的测试用例库；②优化的软件故障模型；③优化的测试策略和测试计划；④新的测试经验和新的测试用例设计思路。

事实上，测试设计过程是循环往复的，并且过程中的每一步骤都可以返回前面的任何一个步骤，即使单独一个测试用例也可能经过以上步骤多次。另外，测试设计的最重要的工作就是设计测试用例。测试用例的衡量标准：多、快、好、省。测试用例库是一种经验积累，是测试活动最宝贵的财富。最后，在系统测试中设计测试用例最常用到的思路是软件攻击。

6.8.5 系统测试手段

测试如同打仗，孙子兵法中的“知己知彼、百战不殆；攻其薄弱、避其锋芒；分而治之、逐个歼灭”作战策略在软件测试，特别是软件的系统测试中是非常具有指导意义的。作战的最主要手段就是进攻或攻击。软件测试中的攻击——软件攻击就是要寻找系统中最容易出现错误的地方进行测试（如寻找开发过程中容易出现疏忽的地方），保证多、快、好、省地找出错误。软件攻击的核心是基于故障模型的测试用例设计。

1. 软件故障模型

故障模型是将测试员的经验和直觉尽量归纳和固化，使得可以重复使用。测试员通过理解软件在做什么，来猜测可能出错的地方，并应用故障模型有目的地使它暴露错误。可以认为故障模型类似于系统设计中的定式（Pattern）思想，具体的攻击方法就是一个一个地定式。因此，对测试员来说，是要能够构造出一个准确的故障模型，使用该故障模型来决定测试策略、测试设计和测试用例运行。

在建立故障模型时，希望故障模型在框架上是通用的，但是建立具体的故障模型时一定要针对具体的软件类型、应用环境甚至开发工具才有意义。

在进行故障模型设计时，要重点考虑软件的行为特性，如软件功能和技术特点（包括输入、输出、数据以及处理等）、软件操作环境（用户界面、文件系统、操作系统环境以及其他软件和操作系统应用编程接口 API 等）。按照这种思路设计的故障模型是一个二维模型，该模型从软件功能和技术特点来说只接收正确的输入并正确地处理，只输出用户接受的并且正确的输出，保持数据结构的完整性（数值、精度和位置）和自我保护的合法计算、功能交互和数据共享；而从软件操作环境来说主要关注用户界面，关注文件系统接口、数据库系统接口，关注资源调配

和管理，以及关注系统 API 调用。

2. 典型攻击方法

基于上述软件行为特性的分析和故障模型设计的考虑，我们重点关注一些典型的攻击方法。这些攻击方法大致从 4 个方面进行考虑：①何时施加攻击，即测试软件的什么功能时适合使用这种攻击，这种攻击针对的功能是什么；②什么样的软件故障会使攻击成功，即本攻击方法主要会暴露实现过程中哪方面的问题，在实现技术上是什么原因产生了这种错误；③如何确定攻击暴露了失效，即本攻击方法成功的标志是什么？需要掌握业务背景知识，或者说理解什么是预期的输出结果；④如何进行攻击，即本攻击方法的操作步骤。

下面以 25 种典型攻击方法为例来进行攻击方法的说明。

1）用户接口输入攻击 6 种

（1）使用非法输入：所有非法输入有错误处理代码吗？代码正确吗？

（2）攻击使用默认值的输入：变量初始化了吗？

（3）使用特殊字符集和数据类型的合法输入：正确处理特殊字符和数据类型了吗？

（4）使用使缓冲区溢出的合法输入：检查字符串/缓冲区的边界了吗？

（5）使用可能产生错误的合法输入组合：输入之间的组合关系考虑吗？

（6）重复输入相同的合法输入序列：循环处理的边界考虑到了吗？

2）用户接口输出攻击 4 种

（1）产生同一个输入的各种可能输出：一个正确的输入在不同情况下产生不同的输出，这些不同情况都考虑充分了吗？

（2）强制产生不符合业务背景知识的无效的输出：开发人员具有解域/业务背景知识吗？是否会产生不符合业务背景的输出？

（3）强制通过输出修改一些属性：初始化代码和修改代码同步吗？

（4）检查屏幕刷新：屏幕刷新时机正确吗？屏幕刷新区域计算正确吗？

3）用户接口数据攻击 3 种

（1）制造使内部数据与输入的组合不相容的情况：处理输入的时候，考虑到内部数据的各种可能的组合了吗？

（2）制造使已有内部数据结构集合溢出的情况：上溢（增加一个元素到集合中）、下溢（删除集合中最后一个元素，或者从空集合中删除元素）。

（3）制造使已有内部数据结构不符合约束的情况：初始化代码和修改代码同步吗？

4）用户接口计算攻击 4 种

（1）使用非法的操作数和操作符组合，攻击用户可以控制计算要求的情况：考虑到操作符和计算要求的合法性了吗？

（2）使函数递归调用自身：考虑到循环/递归的中止了吗？

（3）使计算结果溢出：考虑数据结构是否能够正确存储可能的计算结果了吗？

（4）攻击共享数据或互相依赖的功能计算：一个函数在修改共享数据的时候考虑到其他函数对这个共享数据的假设/约束了吗？

5）文件系统介质攻击 3 种

（1）使文件系统超载：检查文件访问函数的返回值了吗？正确处理文件访问失败的情况了吗？

（2）使介质处于忙或者不可用状态：检查文件/介质访问函数的返回值了吗？正确处理文件/介质访问失败的情况了吗？

(3) 损坏介质(这时候 OS 可能认为介质可用):检查文件/介质访问函数的返回值了吗?正确处理文件/介质访问失败的情况了吗?

6) 文件系统文件攻击 3 种

(1) 使用特殊字符/特殊长度/无效的文件名:处理文件名的代码考虑到各种情况了吗?

(2) 改变文件访问权限:访问文件的函数考虑到文件访问权限了吗?文件访问失败,有正确处理错误的代码吗?

(3) 使文件内容错误,并让系统使用这个文件:检查文件访问函数的返回值了吗?错误处理代码正确吗?

7) 操作系统和软件接口攻击 2 种

(1) 记录-仿真攻击:模拟操作系统和操作环境故障,并记录软件对该故障的反应。如软件正确处理内存故障/网络故障等操作系统和软件接口故障了吗?

(2) 观察-失效攻击:观察底层 API 调用,并动态修改 API 调用,制造错误。如软件正确处理操作系统和软件 API 调用错误的情况了吗?该攻击方法一般用于对可靠性和稳定性要求非常高的软件。

上述的攻击中许多要用到软件攻击工具,如文件系统介质攻击、文件系统文件攻击、操作系统和软件接口攻击等。否则,攻击成本太高:比如物理毁坏介质;工作量太大:比如使用大文件填充硬盘、使用多任务抢占 CPU/网络/内存等资源;有一些不容易实现:动态监控和修改 API 调用。我们一般采用软件故障植入的方法设计软件攻击工具。

3. 软件故障植入

一般软件中的程序代码分为功能代码(通过实现用户需求来完成软件任务)和异常处理代码(通过异常或者其他错误处理机制来处理程序错误),故障植入的目标是强制执行异常处理代码(没有异常处理代码也是一种错误),从而发现其中的错误。

故障植入方法分为编译期植入(在源代码中插入引发故障现象的代码)和运行期植入(在目标代码或运行环境中植入引发故障现象的代码)两类。其中,运行期软件故障植入具有更大的优势:①不需要源代码(因为系统测试阶段经常没有源代码);②可以达到模拟环境故障的目标,如网络中断、网络繁忙、内存匮乏等;③可以以 API 调用失败的方式模拟故障,因为在程序看来,任何环境故障实质上都是一系列的 API 调用失败;④可以只影响被植入的程序,因为使用的是模拟 API 调用失败方式。

运行期软件故障植入可通过截获 API 的方式来实现。一般用到三种方式:①基于调用源截获,找出程序中调用 API 的名字或 DLL,然后用自己的函数替换掉;②路径内截获,如果程序中使用 VTable 等技术间接记录调用 API 地址,则直接修改 VTable 中的 API 地址,用自己的函数替换掉;③目的地截获,修改被调用 API 地址,修改头部代码,转向自己的函数,这实际上采用的是病毒方式。

运行期软件故障植入的实现策略有两种。

(1) 基于定式的故障植入:模拟环境故障,记录故障特征和影响到的 API,修改影响到的 API;采用乱棒打师傅方式的软件攻击法——canned HEAT (Hostile Environment Application Tester),并记录下由此引起的故障产生情况。

(2) 系统的基于调用的故障植入:观察使用的 API,并独立地、细粒度地修改影响任何一个 API;采用偷梁换柱、李代桃僵方式的软件攻击法,并观察因其破坏而引起的失效攻击。

4. 软件攻击的突破口

对系统质量影响最大的地方是系统最薄弱、最容易出问题的地方,这些地方也是软件攻击

的突破口。

1）错误处理测试

健壮性是软件质量的一个重要因素。错误处理测试是检查软件在面对错误时，是否进行了正确的处理。

错误处理测试的目的是要发现软件是否做了用户不期望的事情、发现软件在发生异常的时候是否有能力进行处理。此时，测试人员需要以否定的态度来思考问题。另外，在错误处理测试中发现的部分问题可能不会被修复。

错误处理测试主要考虑典型的异常情况，如用户输入非法数据（不输入数据；输入无效数字数据，如负数和字母数字串；输入任何被认为是非法的数据类型格式；尝试不常用的数据组合；确保使用零值；输入超过或者短于要求长度的数据），在系统不支持的平台上运行，网络连接异常，数据文件（或者数据库）被破坏，数据文件（数据库）中有混乱的数据，计算机断电后启动，在用户界面上的违反操作步骤的操作等。

2）内存泄露测试

内存泄漏是一种典型的程序缺陷，导致应用程序不断消耗系统内存（或虚拟存储器），使程序运行出现响应变慢、某些功能无法实现，甚至整个系统瘫痪等问题。尤其对于嵌入式系统这种资源比较匮乏、应用非常广泛，而且往往又处于重要部位的程序，内存泄漏将可能导致无法预料的重大损失。在某些语言（如 C/C++ 语言）编写的程序中，内存泄露是一个极其普遍的问题。

内存泄露测试可采用静态测试和动态测试技术。首先，通过测量内存使用情况，了解程序内存分配的真实情况，发现对内存不正常的使用；另外，在问题出现前发现征兆，在系统崩溃前发现内存泄露错误；最后，发现内存分配错误，并精确显示发生错误时的上下文情况，指出发生错误的原因。MI Compuware 公司的 BoundChecker、IBM Rational 的 Purify 就是一种典型的内存泄露检查工具。

3）用户界面测试

图形用户界面测试和评估的重点是正确性、可用性和视觉效果，界面中的文字检查和拼写检查也是用户界面测试的重要环节。用户界面测试的过程中，有时要依赖于测试人员的主观判断，但用户界面测试也要遵循一些基本原则，如可用性、规范性、合理性、美观与协调性、菜单位置、独特性、快捷方式的组合、排错性考虑等。表 6-8 给出了界面测试的一些基本考虑。

表 6-8 界面测试指标

指　标	检　查　项	测试人员评价
合适性和正确性	用户界面是否与软件的功能相融洽？	
	是否所有界面元素的文字和状态都正确无误？	
容易理解	对于常用的功能，用户能否不必阅读手册就能使用？	
	是否所有界面元素提供(例如图标)都不会让人误解？	
	是否所有界面元素提供了充分而必要的提示？	
	界面结构能否清晰地反映工作流程？	
	用户是否容易知道自己在界面中的位置，不会迷失方向？	
	有联机帮助吗？	
及时反馈信息	是否提供进度条、动画等反应正在进行的比较耗时间的过程？	
	是否为重要的操作返回必要的结果信息？	

续表

指　标	检　查　项	测试人员评价
出错处理	是否对重要的输入数据进行校验?	
	执行有风险的操作时,有"确认"、"放弃"等提示吗?	
	是否根据用户的权限自动屏蔽某些功能?	
	是否提供 Undo 功能用以撤销不期望的操作?	
风格一致	同类软件的界面元素是否有相同的视感和相同的操作方式?	
	字体是否一致?	
	是否符合广大用户使用同类软件的习惯?	
适应各种水平的用户	所有界面元素都具备充分必要的键盘操作和鼠标操作吗?	
	初学者和专家都有合适的方式操作这个界面吗?	
	色盲或者色弱的用户能正常使用该界面吗?	
国际化	是否使用国际通行的图标和语言?	
	度量单位、日期格式、人的名字等是否符合国际惯例?	
合理布局和谐色彩	界面的布局符合软件的功能逻辑吗?	
	界面元素是否在水平或者垂直方向对齐?	
	界面元素的尺寸是否合理? 行、列的间距是否保持一致?	
	是否恰当地利用窗体和空间的空白,以及分割线条?	
	窗口切换、移动、改变大小时,界面正常吗?	
	界面的色调是否让人感到和谐、满意?	
	重要的对象是否用醒目的色彩表示?	
	色彩使用是否符合行业的习惯?	
个性化	是否具有与众不同的、让用户记忆深刻的界面设计?	
	是否在具备必要的"一致性"的前提下突出"个性化"设计?	

4）性能测试

性能测试包含并发性能测试、强度测试、破坏性测试等方面。

并发性能测试是评估系统交易或业务在渐增式并发情况下处理瓶颈以及能够接收业务的性能过程。

强度测试是在资源受限的情况下,找出因资源不足或资源争用而导致的错误。

破坏性测试重点关注超出系统正常负荷若干倍的情况下,错误出现状态和出现比率以及错误的恢复能力。

性能测试可以通过"黑盒"测试或者"白盒"测试方法来进行,一般要借助工具。如 IBM Rational Performance Tester、Compuware QALoad 以及 HP Loadrunner 等。

在进行性能测试时,要求:①测试程序在获得定量结果时程序计算的精确性;②测试程序在有速度要求时完成功能的时间;③测试程序完成功能所能处理的数据量;④测试程序各部分的协调性,如高速、低速操作的协调;⑤测试软/硬件中哪些因素限制了程序的性能;⑥测试程序的负载潜力;⑦测试程序运行占用空间。

性能测试应用场合有:①软件中某个模块涉及复杂的计算,特别是一些基于人工智能的分析;②涉及大量数据的读写、通信;③涉及数据检索,而被检索的数据,具有很大的数据量;④具有多个并发用户;⑤软件在运行时,可用资源(特别是 CPU 和内存)可能在某些情况下很紧张。例如一些嵌入式系统软件。

5）压力测试

压力测试也叫负荷测试,即获取系统能正常运行的极限状态。压力测试用于检查软件在

面对大数据量时是否可以正常运行大数据量，往往是发生概率比较小的情况。

压力测试所涉及的方面主要包括数据库大小、磁盘空间、可用内存空间、数据通信量。表 6-9 给出了压力测试的测试模板。

表 6-9　压力测试模板

极限名称 A	如“最大并发用户数量”	
前提条件		
输入/动作	输出/响应	是否能正常运行
如 10 个用户并发操作		
如 100 个用户并发操作		

6）软件运行时错误

运行时错误是指应用程序在运行期间执行了非法操作或某些操作失败时出现的错误，是所有的软件错误中最具风险的。图 6-29～图 6-31 说明了软件运行时错误产生的例子。



图 6-29　Windows 2000 操作系统上发生运行时错误的现象

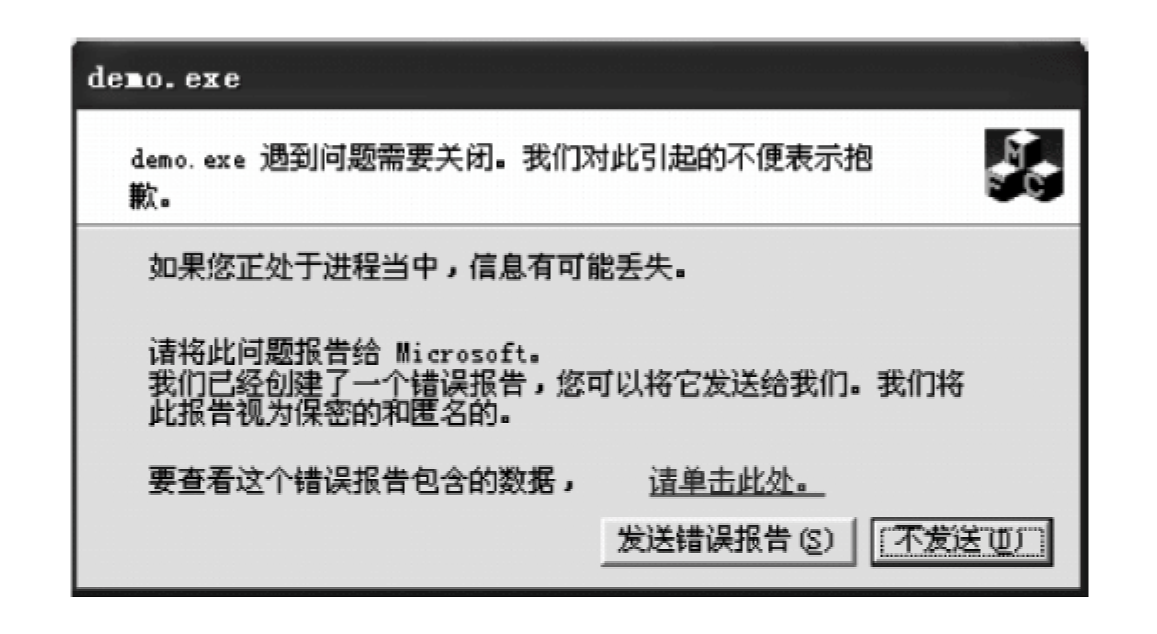


图 6-30　Windows XP 操作系统上发生运行时错误的现象

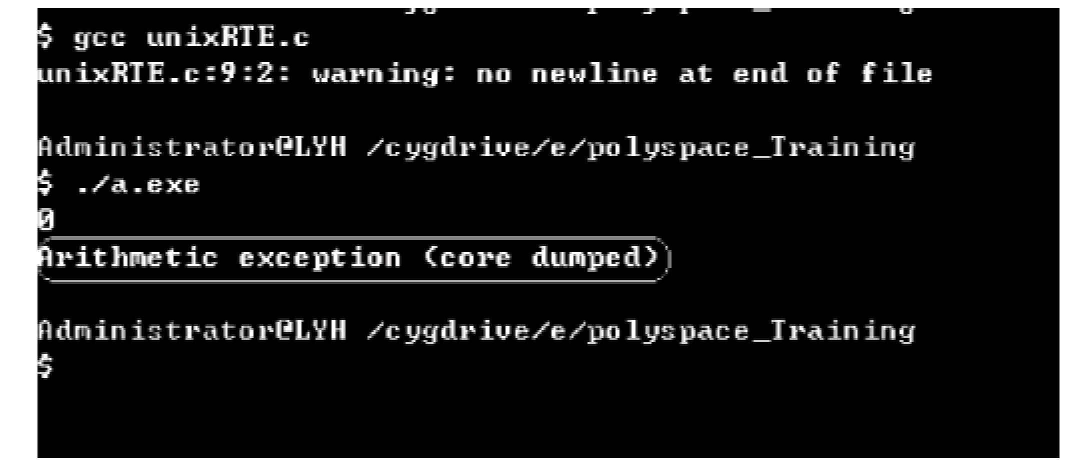


图 6-31　UNIX 操作系统上发生运行时错误的现象

在这种情况下，不管我们做什么选择，应用程序都会退出。可能对于一般的软件来说，出现这样的错误没关系，但对于航空航天、汽车以及医疗设备等安全级别要求非常高的系统来说，一旦出现这样的运行错误，损失就是不可估量的。因此，对于关键或重要的软件必须进行运行时检查。

运行时检查是实际运行时检测程序的方案，通过选择适当的测试用例，并对程序的运行状态进行监控以发现程序中的错误。该技术依靠系统编译程序和动态检查工具实现检测，优点是检测的结果比较接近于程序的真实运行状态，对于内存使用情况、空指针引用等错误的检测比较准确。但缺点是检测的全面性严重依赖于测试用例对代码的覆盖情况，运行时间较长，而且对于程序中的递归调用等过程，动态检测很难覆盖全部的情况。

7）回归测试

回归测试是指对某些已经被测试过的内容进行重新测试，如软件增加后影响软件的结构，

软件修改考虑不周而引入问题。

回归测试的目标有两个：①检查测试出的软件问题是否得到了正确的修改；②保证被测软件在被修改之后，各项功能依然正确(未引入新的缺陷)。

一般来说，应该对修改的部分进行针对性的测试保证其符合需求，随后还要运行一系列的测试确认现有功能仍然符合需求。

在工程实践中使用的回归测试有多种策略：①通过进行用例的相关性分析，找到与软件修改部分相关的测试用例进行测试；②在不进行任何分析的前提下对测试用例的全集进行测试；③必要时可能会生成新的测试用例专门用来进行回归测试；④测试人员可按实际情况选择回归策略，如每两周需要进行一次完整的回归测试。或当修复的缺陷数量累计到 50 个时，进行一次完整的回归测试。也可以在产品递交用户前 5 个工作日，进行完整的回归测试。

回归测试通常要借助于自动化测试工具。

6.9 动态测试工具介绍

6.9.1 国产单元测试工具 Visual Unit

广州凯乐公司研发的 Visual Unit(当前最新版本为 3.0)，简称 VU，是新一代单元测试工具，功能强大，使用简单，是完全可视化、自动化的 C/C++ 单元测试工具，具有自动打桩(包括补齐、隔离、控制)、自动生成测试代码和用例框架、可视化编辑测试用例等特性。VU 自动生成测试代码功能，使单元测试人员不需花费时间、中断思路去编写测试代码。

VU 充分地显示出代码的行为：显示各种数据的输入输出值，显示不同输入时程序所执行的代码；画出逻辑结构图及不同输入时程序的执行路径，程序员可以随时浏览自己的劳动成果。同时又提高了编程的效率，总体来说，边编码边用 VU 进行测试，在达到完整测试的同时，还能大幅度减少开发时间。

VU 有助于测试的完整性，使用 VU，在“白盒”测试方面，易于达到 100%语句、条件、分支、路径覆盖，提供详尽的测试报告和待测试文件列表，随时可以检验测试效果、找出遗漏代码或未完成覆盖的代码，保证测试的完整性；在“黑盒”测试方面，可以轻松完成功能测试、边界测试、速度测试等。这种测试完整性，使代码中的缺陷尽量显现。在回归测试方面，VU 随时可以用回归测试检验修改是否引入新的错误，因此，随时可以对项目的设计进行或大或小的修改，轻松进行螺旋式的迭代开发，或边开发边设计，甚至“以开发代替设计”，使项目或产品真正符合用户的需求。

VU 还帮助程序员快速地排除错误和高效地调试，尽可能减少程序员查找某种错误的时间，使程序员的思维始终集中在程序逻辑上。VU 所具有的增强调试器功能(如自由后退、用例切换)，大大地提高了调试的效率。

VU 不仅是单元测试工具，更是一种使程序开发变得更高质、更高效、更舒适的工具。VU 的测试结果使程序行为一目了然，有助于整理编程思路，提高编程效率和正确性，并能快速排错。VU 目前版本适用于 C++语言。

在现有开发队伍和管理水平的基础上，使用 VU 进行充分的单元测试，可以使项目或产品的质量较大幅度地提高，同时开发成本还要较大幅度地下降。

VU 功能强大但使用简单，学习资料丰富，一天时间就能轻松上手；在 VU 的支持下编程，程序行为一目了然，感觉舒适有趣，很受程序员的喜爱，决不会受到开发人员的抵制；VU

提供详尽的测试报告，测试部门可以依据测试报告对测试结果进行审核，保证测试质量。

下面以 Visual Unit 2.0 为例具体地介绍其主要功能。

1. 界面总览

界面主要有三个视图：函数视图、类/文件视图、全景视图，每个视图八到十页。在左边函数树中单击函数名，显示相应的函数视图；选择一个类，显示相应的类/文件视图；单击 All，显示全景视图。

2. UDT 工程（单元开发与测试工程）

UDT（Unit Development Test，单元开发与测试工程）工程解决了并行开发难题和高耦合代码难于分割测试的难题。一个项目可以建立任意数量的 UDT 工程。

UDT 工程从项目（称为主项目）中切割出“一块”开发或测试任务，以便分配给一位成员进行开发或测试。VU 通过补齐（自动生成未定义符号）、隔离（自动生成桩代码替换部分源代码）等技术手段，建立可单独编译链接的产品子工程（分离自主项目的部分代码及自动生成的桩代码，称为子项目）和测试工程（VU 自动生成的测试代码），可以脱离主项目进行开发、测试，或边开发边测试。

如何指定切割目标和范围呢？通过设定被测源文件、外围源文件来实现。被测源文件是开发或测试标的；外围源文件是指开发或测试过程中需要使用，但不测试的源文件；其他源文件称为隔离源文件，此外，头文件还可指定是否用于引入库（引入静态或动态库的头文件，符号已在库中实现，因此不需要生成桩代码）。每种文件均可选择多个目录，并可精确指定目录下每个文件的类别。

3. 用例编辑

VU 自动生成测试代码及边界测试用例。由于工具不可能自动了解被测代码的功能，所以普通用例的输入输出由人工定义。VU 会生成第一个用例的框架，简单的输入输出可直接填写数值，复杂输入输出可以使用任何 C/C++代码，可以切换到代码模式或在开发环境中编辑用例代码。单击“新建”，VU 会自动复制当前用例，修改一两个输入输出即可获得新用例。

4. 桩控制

无须编写代码，可在用例中随意控制子函数的行为，不仅可以指定返回值，还可以设定输出参数、成员变量、全局变量的值。这些“值”可以是任意类型，并且多次调用同一子函数还可以设定不同行为。可以自动判断子函数是否执行及执行次数是否符合预期。在用例助手中，双击子函数名或参数，VU 就会弹出“生成桩控制代码”对话框并填好初始数据。

5. 测试输出

测试输出不但显示出测试是否通过，还统计和标示语句、条件、分支、路径覆盖，并自动打印输入输出数据和标示用例执行的代码，程序行为一目了然。测试输出不但能快速找出程序错误，边开发边测试还能帮助整理和检验编程思路，提高开发效率。单击“虫”图标，即可启动调试。

6. 用例设计

用例设计器用于实现完整的白盒覆盖。选中未覆盖的语句、条件、分支或路径，打开用例设计器，VU 就会从现有用例中计算出一个近似用例（近似是指所需修改最少），并生成修改提示，按提示修改近似用例，即可覆盖预期的逻辑目标。用例设计器使实现 100%语句、条件、分支、路径覆盖不再困难。

7. 测试统计

自动统计未测、已测、错误（含有失败的测试）、欠缺（语句、条件、分支或路径至少有一项未

实现 100%覆盖)的函数,可查看全部或一个类/文件的统计数据。

8. 测试报告

自动生成 HTML 格式的测试报告,测试报告可在 VU 中浏览,也可以导出或复制到其他电脑上,用普通浏览器浏览。

VU 3.0 在易用性、用户体验上是一个飞跃。主要更新有:①集成了 IDE 的基本功能,测试过程不再使用测试 IDE。②全面优化了性能,大幅提高了解析速度,降低了内存占用。③更换了界面图标及产品 LOGO。④文件改用短路径,一是便于工程的迁移,二是消除了路径过深或过长产生的问题。⑤增加了自动统计最近更新函数的功能。在开发过程中,如果修改了多个函数,可以使用此功能执行修改过的函数的测试。⑥优化了复杂类型参数的底层模拟。⑦一系列的小改进。

VU 的试用版本及免费版本可在凯乐公司的网站(http://www.kailesoft.cn)下载。

6.9.2 开源集成测试工具 Selenium

Selenium(SeleniumHQ)是 Thoughtworks 公司的一个集成测试的强大工具,它是一个用于 Web 应用程序测试的工具。Selenium 测试直接运行在浏览器中,就像真正的用户在操作一样。支持的浏览器包括 IE、Mozilla 和 Firefox 等。这个工具的主要功能包括:①测试与浏览器的兼容性(测试应用程序是否能够很好地工作在不同浏览器和操作系统之上);②测试系统功能。

Selenium 现在存在两个版本,一个叫 Selenium-Core,一个叫 Selenium-RC。

Selenium-Core 是使用 HTML 的方式来编写测试脚本,但也可以使用 Selenium-IDE 来录制脚本,但是目前 Selenium-IDE 只有 FireFox 版本。

Selenium-RC 是 Selenium-Remote Control 缩写,是使用具体的语言来编写测试类。

Selenium 能被选为最好的集成测试、回归测试方案,是因为:①Selenium IDE,一个 FireFox 插件,能自动记录用户的操作,生成测试脚本。②生成的测试脚本可以用 Selenium Core 手工执行,也能基于 Selenium RC 放入 Java、C#、Ruby 的单元测试用例中自动运行。③测试用例调用实际的浏览器(如 IE、FireFox)来执行测试。和有些开源方案自行实现 Web 解释引擎相比,实际的浏览器能模拟更多用户使用,顺便还可以测试各浏览器兼容性。④测试脚本语法非常简单。

Selenium 的最新版本为 2.33.0,下载地址:http://docs.seleniumhq.org/download。

图 6-32 是 Selenium IDE 的运行界面截图。

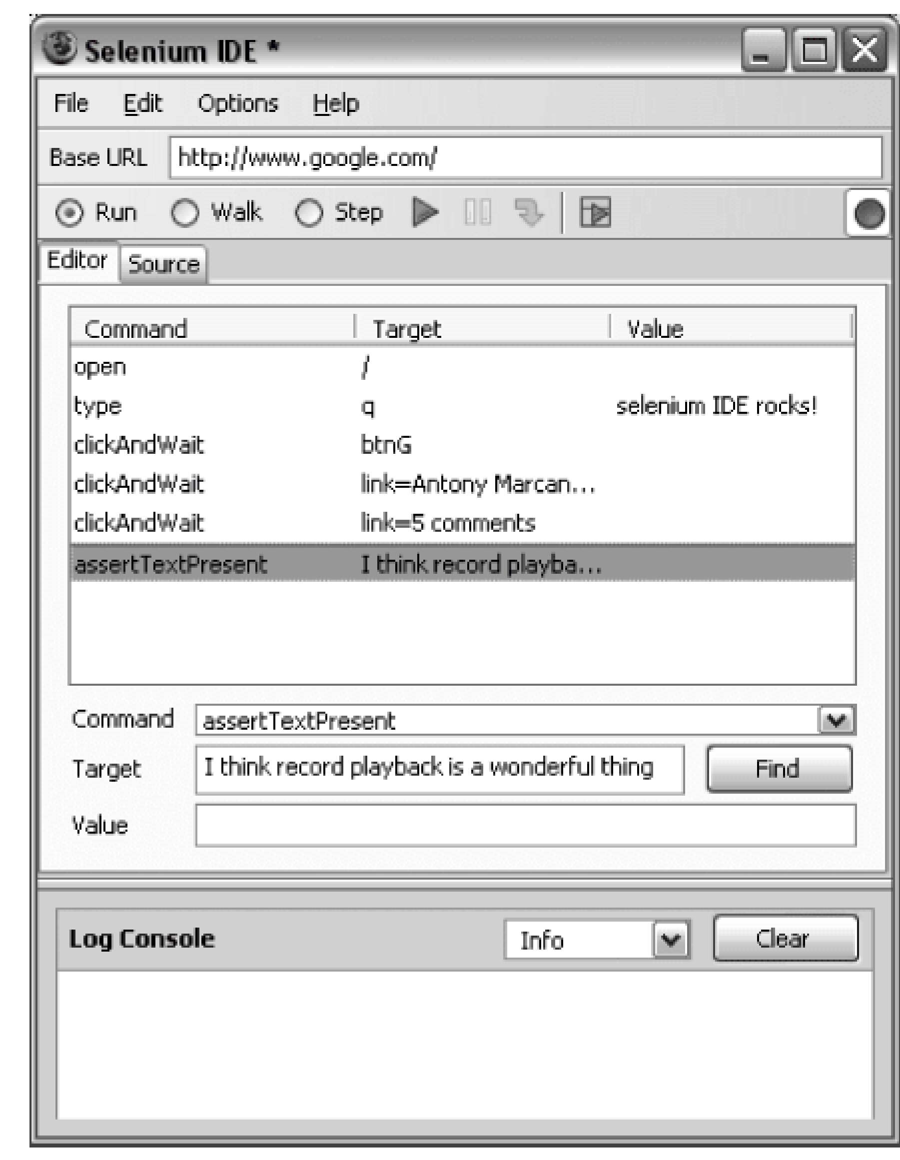


图 6-32 Selenium IDE 界面

6.9.3 系统测试工具

目前用于系统测试的工具主要有功能测试工具、性能测试工具和安全测试工具。

1. 功能测试工具

这里提到的系统功能测试工具主要是针对有界面的应用而言的。这些系统功能测试工具通过自动录制、检测和回放用户的应用操作,将被测系统的输出记录同预先给定的标准结果进

行比较,以发现不一致的地方。

1）惠普功能测试工具 QuickTest Professional(简称 QTP)

QTP 是一种企业级的用于检验应用程序是否如期运行的功能性测试工具。通过捕获、检测和重复用户交互的操作,能够辨认缺陷并且确保那些跨越多个应用程序和数据库的业务流程在初次发布就能避免出现故障,并且保持长期可靠运行。

QTP 可以覆盖绝大多数的软件开发技术,简单高效,并具备测试用例可重用的特点。

2）QARun

为当今关键的客户/服务器、电子商务到企业资源规划(ERP)应用提供企业级的功能测试。通过测试脚本开发和测试执行自动化,QARun 帮助测试人员和 QA 管理人员更有效地工作以加快应用开发。

QARun 适用于所有关键业务应用测试,它可以在复杂的企业环境里测试各种各样的应用。QARun 与 QTP 相比学习成本要低很多。不过要安装 QARun 必须安装.NET 环境,另外它还提供与 TestTrack Pro 的集成。

3）IBM Rational Robot

IBM Rational Robot 是人们经常使用的功能测试工具,属于 IBM Rational TestSuite 中的一员,对于 Visual studio 编写的程序支持得非常好,同时还支持 Java Applet、HTML、Oracle Forms、People Tools 应用程序。要支持 Delphi 程序的测试还必须下载插件。IBM Rational Robot 使用 Basic 语法,使用 SQABasic 语言。

2. 性能测试工具

性能测试工具的主要目的是度量应用系统的可扩展性和性能。在性能测试过程中,通过实时性能监测来确认和查找问题,并针对所发现问题对系统性能进行优化,确保应用的成功部署。性能测试工具能够对整个企业架构进行测试,通过这些测试,企业能最大限度地缩短测试时间,优化性能和加速应用系统的发布周期。

1）HP LoadRunner

HP LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具。通过模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题,LoadRunner 能够对整个企业架构进行测试。

LoadRunner 是一种适用于各种体系架构的自动负载测试工具,它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统,它通过模拟实际用户的操作行为和实行实时性能监测,来帮助我们更快地查找和发现问题。此外,LoadRunner 能支持广泛的协议和技术,为我们的特殊环境提供特殊的解决方案。

2）QALoad

QAload 是 Compuware 公司性能测试工具套件中的压力负载工具,QALoad 是客户/服务器系统、企业资源配置(ERP)和电子商务应用的自动化负载测试工具。QALoad 可以模拟成百上千的用户并发执行关键业务而完成对应用程序的测试,并针对所发现问题对系统性能进行优化,确保应用的成功部署,其主要功能有:①预测系统性能;②通过重复测试寻找瓶颈问题;③从控制中心管理全局负载测试;④快速创建仿真的负载测试;⑤集成的系统资源视图。

QALoad 除了测试 Web 应用外,还可以测试一些后台的东西,如 SQL Server 等。只要它支持的协议,都可以测试。

3）JMeter

开源性能测试工具 JMeter 是一个专门为服务器负载测试而设计、100%的纯 Java 桌面运

行程序。早期它是为 Web/HTTP 测试而设计的,但是它已经扩展到支持各种各样的测试模块。它与 HTTP 及 SQL(使用 JDBC)的模块一起运行。它可以用来测试静止或活动资料库中的服务器运行情况,可以用来模拟服务器或网络系统在重负载下的运行情况。它也提供了一个可替换的界面用来定制数据显示、测试同步及测试的创建和执行。

习题

1. 什么是动态测试? 动态测试包括哪些内容?
2. 什么是“白盒”测试? “白盒”测试采用哪些方法? 如何进行“白盒”测试?
3. 什么是逻辑覆盖、路径测试? 它们包含哪些内容?
4. 什么是数据流测试? 如何进行数据流测试?
5. 什么是信息流测试? 信息流测试包含哪些内容?
6. 什么是“黑盒”测试? “黑盒”测试一般采用哪些方法? 如何进行“黑盒”测试?
7. 什么是等价类划分法、边界值分析法? 如何应用这些方法进行“黑盒”测试?
8. 什么是因果图分析法? 如何用因果图生成测试用例?
9. 什么是“灰盒”测试? 如何开展“灰盒”测试?
10. 测试用例设计的原则和要素是什么? 如何进行测试用例的设计?
11. 简述测试用例分级及测试用例优先级的概念。
12. 什么是单元测试? 单元测试一般包含哪些内容? 如何进行单元测试?
13. 什么是集成测试? 集成测试采用的方法有哪些? 进行集成测试时我们要考虑哪些问题?
14. 简述确认测试的概念。确认测试包括哪些内容。
15. 什么是系统测试? 系统测试包括哪些内容? 如何进行系统测试?
16. 系统测试中我们常关注的测试类型是哪些? 为什么?
17. 什么是回归测试? 它有什么用处? 一般如何进行回归测试?

第3部分

软件测试管理方法与技术篇

软件测试工作不仅要有计划地进行，而且需要科学地组织和管理，这样才能开发出高质量的软件产品。对测试活动进行组织策划和有效管理，才能使软件测试在软件质量体系保障中发挥应有的重要作用。

软件测试管理是软件项目管理的一个子集或分支，与传统的软件项目管理在核心上没有本质的区别，但由于管理对象的特殊性，因此在具体执行上具有较大的不同。

为改进软件测试的质量，管理组织必须按照6个步骤进行：①理解当前的测试过程的状态；②开发希望的过程；③按照优先级列出需要改进的活动；④为这些改进活动制定计划；⑤承诺执行活动所需的资源；⑥重新开始步骤①。而这些都属于软件测试过程管理中的内容。

另外，软件测试是为了尽可能多地发现软件中的缺陷并将其修复，从而提高软件整体质量。事实上，每一个软件组织都知道必须妥善处理软件中的缺陷。这是关系到软件组织生存、发展的质量根本。

综上所述，软件测试过程管理和缺陷管理是软件测试管理的核心内容。

第 7 章 软件缺陷与缺陷管理

软件危机最重要的特征就是：软件产品的质量靠不住，错误一大堆，难以维护。特别是大型软件，其错误更多，维护更加困难。因此，为了保证软件正常运行，必须对软件中存在的缺陷进行检查或测试，对发现的错误进行有效的管理，从而为软件缺陷或错误的消除或者软件质量的评价及软件开发的决策提供依据。

7.1 软件缺陷

1991 年海湾战争中，美军使用爱国者导弹拦截伊拉克飞毛腿导弹，出现过几次拦截失败事件，后经查明是软件计时系统的累积误差所致，该软件缺陷是一个很小的系统时钟错误积累起来造成十几个小时的延迟，致使跟踪系统准确度丧失，最终导致了严重的后果。事实上，软件系统的可靠性是很难保证的，几乎没有不存在错误或缺陷的软件系统。这是因为，软件错误或软件缺陷是软件产品的固有成分，是软件"生来具有"的特征。不管是小程序还是大型软件系统，无一例外地都存在缺陷，这些软件缺陷，有的容易表现出来，有的隐藏很深难以发现，有的对使用影响轻微，有的会造成财产甚至生命的巨大损失。

7.1.1 软件缺陷定义

1. 什么是软件缺陷

在需求分析和设计过程中，需求及需求规格说明在某种程度上与用户要求不符，或者设计中存在一些错误；在写完程序进行编译时会出现语法错误、拼写错误或者程序语句错误；软件完成后，应有的功能不能使用；或者软件在交付使用后，出现了一些在测试时没有发现的问题，造成软件故障等，所有这些都可以称为软件的缺陷，可以用图 7-1 表示。

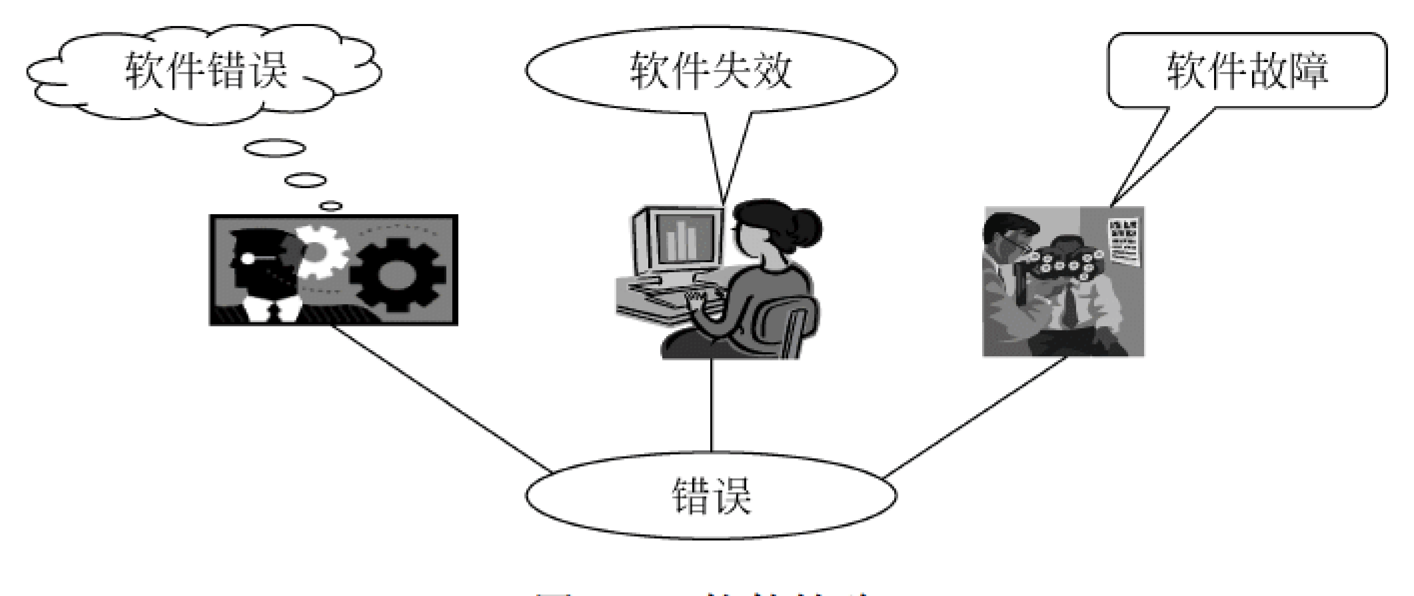


图 7-1　软件缺陷

软件缺陷包括检测缺陷和残留缺陷。检测缺陷是指软件在用户使用之前被检测出的缺陷；残留缺陷是指软件发布后存在的缺陷，包括在用户安装前未被检测出的缺陷以及检测出

但未被修复的缺陷。用户使用软件时，因残留缺陷引起的软件失效症状称为软件故障。软件故障是软件没有表现出人们所期待的正确的结果。软件失效是指软件出现的一些状态，如：①功能部件执行其功能的能力的丧失；②系统或系统部件丧失了在规定的限度内执行所要求功能的能力；③程序操作背离了程序需求。缺陷是这些故障和失效的源头，要想获得高质量的软件就要从消除软件缺陷着手，进行缺陷的预防、检测和消除工作。

软件缺陷简单说就是存在于软件(文档、数据、程序)之中的那些不希望，或不可接受的偏差，而导致软件产生的质量问题。按照一般的定义，只要符合这 5 个规则中的一个，就判定其存在软件缺陷：①软件未实现产品说明书要求的功能；②软件出现了产品说明书指明不应该出现的错误；③软件实现了产品说明书未提到的功能；④软件未实现产品说明书虽未明确提及但应该实现的目标；⑤软件难以理解、不易使用、运行缓慢或者——从测试员的角度看——最终用户会认为不好。

总之，软件缺陷(Defect 或 Bug)是软件开发过程中的“副产品”，会导致软件产品在某种程度上不能满足用户的需要，导致对软件产品预期属性的偏离，造成用户使用的不便。SW-CMM 对其定义是：“系统或系统成分中能造成它们无法实现其被要求的功能的缺点。如果在执行过程中遇到缺陷，它可能导致系统的失效”。

2. 软件缺陷带来的风险

软件缺陷会为系统带来一系列的风险，试想：①如果软件某些代码产生了错误会导致什么样的结果；②未被验证的数据交换如果被接受又会带来怎样的结果；③如果文件的完整性被破坏，那么是否还能保证系统符合用户的要求，软件是否还能如期按要求交付使用；④如果系统发生严重故障时，如果不能保证系统被安全恢复(完成恢复成被备份时的状态)，甚至导致系统完全崩溃；⑤因某种需求，要暂停系统的运行，而系统又没有办法停止，带来的后果可想而知；⑥进行维护工作时，系统性能下降到不能接受的水平，从而使得用户的需求得不到满足，甚至因此带来严重的影响；⑦系统的安全性是否有保证，尤其是对于一些特殊的系统，若安全性得不到保障，那么这个软件可以说是不合格的；⑧系统的操作流程是否符合用户的组织策略和长远规划；⑨若系统的运行不可靠、不稳定、不易使用和不便于维护，将对用户造成多大的伤害；⑩每一个系统都不是封闭、完全独立的，若系统不能与其他系统相连或者很难与其他系统相连，那么这样的产品也是有问题的。总之，上述其中的任何一点都会造成软件开发的失败，例如，软件不能正常使用，引起灾难性的事件，等等。

3. 软件缺陷产生的原因

现在我们知道了什么是软件缺陷，也了解了软件缺陷的影响，但是软件缺陷为什么会出现呢？事实上导致软件产生缺陷有 9 类原因：①不完善的需求定义；②客户——开发者通信失败；③对软件需求的故意偏离；④逻辑设计错误；⑤编码错误；⑥不符合文档编制与编码规定；⑦测试过程不足；⑧规程错误；⑨文档编制错误。

经过针对上述 9 类原因长时间的调查研究，得到了一个令人惊讶的结论：大多数软件缺陷并不是由于编码造成的，导致大多数软件缺陷产生的最大的原因是需求分析，其次是在软件设计，如图 7-2 所示。

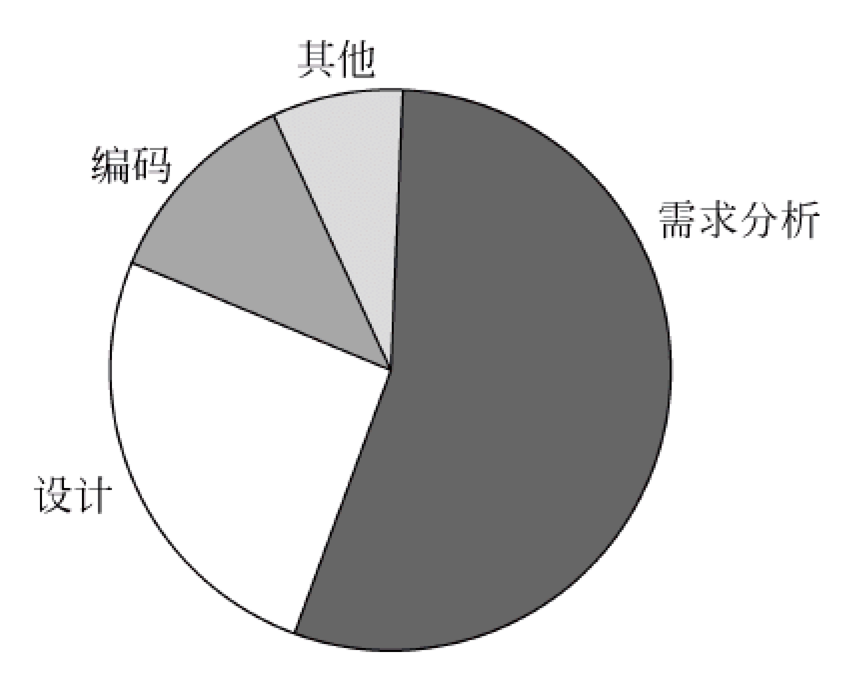


图 7-2 软件缺陷产生的原因很多，其中主要原因是需求分析

1) 需求分析导致缺陷产生的原因

需求分析(也可以说成是需求规格说明)成为造成软件缺陷最大的来源是有原因的。软件需求规格说明

书描述了系统应该具有哪些功能，不应该具有哪些功能，功能的操作性如何，性能如何等具体规格。它是开发初期最重要的过程文档，也是后期开发与测试的重要依据，可以说是开发流程与测试流程的输入。根据过程理论，“正确的输入，正确的过程，正确的解决方案将会产生正确的结果”，如果一开始输入就不正确，那么经过过程的处理后，缺陷/错误会被放大，同时修复的成本会显著上升，人力、物力、时间将会被大量耗费，所以从早期就开始对需求规格说明书进行审查并基线化是必需的。同时测试人员在需求基线化前应该参与到该流程中，参与评审，尽早从客户/测试的角度找出所有不合理/不明确/不可行的需求，减少后期的开发与测试成本。测试人员以及质量人员在开发初期是比较重要的角色，责任比较重大，应当负起责任。由此可见，需求规格说明书是何等重要的文档。

另外，在软件开发之初，由于客户——开发者之间的通信失败，造成需求规格说明的不完善或者是对软件需求的偏离。而后在开发过程中因需求规格说明的不全面或经常变更，再加上整个开发小组不能很好地沟通，造成设计和编码与需求规格说明之间的不一致等。

2）设计导致缺陷产生的原因

设计是另一个缺陷产生的主要来源，设计是软件开发人员规划软件的过程，就好比建筑师在建筑物建造之前要绘制建筑蓝图，在这个过程中可能会存在一些逻辑错误。

另外，设计也会产生变化，会修改，再加上整个开发小组不能很好地沟通，所有这些就导致了软件缺陷的产生。

3）编码导致缺陷产生的原因

软件缺陷在编码阶段出现，通常是由于软件复杂、文档不足、进度压力、普通的低级错误或者是由程序员的思维定式等造成代码出现错误。在编码完成后，软件出现错误，经常听到程序员说：“这是按要求做的，若是有人早告诉我，我肯定就不会这样编写了。”因此，许多软件缺陷貌似是在软件编码阶段出现的，但实际上却是由需求规格说明或软件设计所造成的。

剩下的原因可以归为一类。其可能是由测试错误引起，也可能是因为对需求的理解错误等，但是这部分只占极小的比例。

4. 为什么软件缺陷难以测试或发现

由上可知软件测试人员的任务之一就是尽可能早地找出软件缺陷，并确保其得以修复。现在软件测试已经在软件开发过程中占很大的比重，软件开发人员也越来越重视软件质量，但是为什么仍旧存在那么多的软件缺陷不能被测试到呢？

由于软件是由人来完成的，在目前的技术上不能避免人为因素的错误，以致有错是软件的属性，这是目前改变不了的。现在人们已经逐步认识到所谓的软件危机实际上仅是一种状况，那就是软件中有错误/缺陷，正是这些错误/缺陷导致了软件开发在成本、进度和质量上的失控。因此，必须面对现实，避免软件中错误/缺陷的产生，努力消除已经产生的错误/缺陷，使程序中的错误/缺陷达到尽可能少的程度。

但是，由于软件所具有的特性，使得我们很难找出或排除软件中的错误/缺陷，因为：①软件错误/缺陷很难看到；②软件错误/缺陷看到了但很难抓到；③软件错误/缺陷抓到了但无法修改或很难修改；④人们无时无刻都可能犯错误，使得软件中存在错误/缺陷。

典型的软件错误/缺陷类型有需求定义错误、需求解释错误、需求记录错误、设计说明错误、编码说明错误、程序代码编写错误、数据输入错误、测试错误、问题修改错误、正确的结果是由于其他的缺陷产生的。

因此，尽管软件测试的目的是尽可能多地发现软件中潜在的缺陷，但并不能保证所有的缺陷都被发现。有些缺陷在测试过程中是很难被发现的。

另外，有些软件缺陷虽然在测试过程中可以被发现，但是测试人员不能重现该缺陷，也就是看得见抓不到，从而使得缺陷不能得到修复。

最后，尽管原则上软件缺陷在任何时候都必须得到修复。但由于没有足够的时间、不算真正的软件缺陷、修复的风险太大等原因，产品开发小组可以决定对一些软件缺陷不作修复。

7.1.2 软件缺陷描述

从软件缺陷的定义中我们可以知道判断一个缺陷的唯一标准就是看其是否符合用户的需求。在大型软件开发过程中，会出现成千上万或更多的软件缺陷，要确定这些缺陷怎样描述、分类和跟踪或监控，以确保每个被发现的缺陷都能够及时得到处理。如：确保每个被发现的缺陷都能够被解决；收集缺陷数据并根据缺陷趋势曲线识别测试过程的阶段；收集缺陷数据并进行数据分析，作为组织的过程财富。

在一个运行良好的组织中，缺陷数据的收集和分析是很重要的，从缺陷数据中可以得到很多与软件质量相关的数据。

对缺陷进行跟踪、监控或管理的基本流程是：首先要准确地描述缺陷，其次对各种缺陷进行分类。在这之中，通过对缺陷进行分类，可以迅速找出哪一类缺陷的问题最大，怎样集中精力预防和排除这一类缺陷，并在这几类缺陷得到控制的基础上，再进一步找新的容易引起问题的其他几类缺陷。

对软件缺陷进行有效描述涉及如下内容。

1. 可追踪信息

缺陷 ID(唯一的缺陷 ID，可以根据该 ID 追踪缺陷)。

2. 缺陷基本信息

缺陷的基本信息有如下几部分内容。

(1) 缺陷标题：描述缺陷的标题。

(2) 缺陷的严重程度：描述缺陷的严重程度。一般分为“致命”、“严重”、“一般”、“建议”四种。

(3) 缺陷的紧急程度：描述缺陷处理的紧急程度。从 1 至 4，1 是优先级最高的等级，4 是优先级最低的等级。

(4) 缺陷提交人：缺陷提交人的名字(含邮件地址)。

(5) 缺陷提交时间：缺陷提交的时间。

(6) 缺陷所属项目/模块：缺陷所属的项目和模块，最好能较精确地定位至模块。

(7) 缺陷指定解决人：缺陷指定的解决人，在缺陷“提交”状态为空，在缺陷“分发”状态下由项目经理指定相关开发人员修改。

(8) 缺陷指定解决时间：项目经理指定的开发人员修改此缺陷的截止时间。

(9) 缺陷处理人：最终处理缺陷的处理人。

(10) 缺陷处理结果描述：对处理结果的描述，如果对代码进行了修改，则要求在此处体现出修改。

(11) 缺陷处理时间：对缺陷进行处理的时间。

(12) 缺陷验证人：对被处理缺陷验证的验证人。

(13) 缺陷验证结果描述：对验证结果的描述(包括通过、不通过的结论)。

(14) 缺陷验证时间：对缺陷进行验证的时间。

3. 缺陷的详细描述

对缺陷描述的详细程度直接影响开发人员对缺陷的修改，描述应该尽可能详细。

4. 测试环境说明

对测试环境的描述。

5. 必要的附件

对于某些文字很难表达清楚的缺陷,使用图片等附件是必要的。

6. 从统计的角度出发

从统计的角度出发,还可以添加上"缺陷引入阶段"、"缺陷修正工作量"等项目。

软件缺陷的描述是后面要论述的软件缺陷报告的基础部分,也是测试人员就一个软件问题与开发小组交流的最初且最好的机会。一个好的描述,需要使用简单的、准确的、专业的语言来抓住缺陷的本质。否则,它就会使信息含糊不清,可能会误导开发人员。

清晰准确的软件缺陷描述可以减少软件缺陷从开发人员返回的数量,提高软件缺陷修复的速度,使每一个小组能够有效工作以提高测试人员的信任度;同时,也可以得到开发人员对清晰的软件缺陷描述有效的响应,加强开发人员、测试人员和管理人员的协同工作。

7.1.3 软件缺陷分类

对软件缺陷进行分类,分析产生各类缺陷的软件过程原因,总结在开发软件过程中不同软件缺陷出现的频度,制定对应的软件过程管理与技术两方面的改进措施,是提高软件组织的生产能力和软件质量的重要手段。

1. 软件缺陷分类方法

缺陷分类是在缺陷描述的基础上进行的。在对缺陷进行分类之前,我们首先要定义缺陷的属性,即属性名称描述、缺陷标识(标记某个缺陷的一组符号,每个缺陷必须有一个唯一的标识)、缺陷类型(根据缺陷的自然属性划分的缺陷种类)、缺陷严重程度(因缺陷引起的故障对软件产品的影响程度)、缺陷优先级(缺陷必须被修复的紧急程度)、缺陷状态(缺陷的跟踪修复过程的进展情况)、缺陷起源(缺陷引起的故障或事件第一次被检测到的阶段)、缺陷来源(引起缺陷的起因)、缺陷根源(发生错误的根本因素)。

软件缺陷的分类方法繁多。各种分类方法的目的不同,观察问题的角度和复杂程度也不一样。下面是几个有代表性的软件分类方法。

1) Putnam 分类法

Putnam 等人提出的分类方法将软件缺陷分为六类:需求缺陷、设计缺陷、文档缺陷、算法缺陷、界面缺陷和性能缺陷。

2) 国军标分类法

我国国家军用标准 GJB 437 根据军用软件错误的来源将软件错误分为三类:①程序错误,运行程序与相应的文档不一致,而文档是正确的;②文档错误,运行程序与相应的文档不一致,而程序是正确的;③设计错误,虽然运行程序与相应的文档一致,但是存在设计缺陷,可能产生错误。

该类分类方法可以分析软件缺陷的来源和出处,指明修复缺陷的努力方向,为软件开发过程各项活动的改进提供线索。分类简单是该分类方法的显著特点。因为分类方法简单,所以它提供的缺陷相关信息对具体的缺陷修复工作其贡献或作用有限。

3) Thayer 分类法

Thayer 软件错误分类方法是根据错误性质分类,它利用测试人员在软件测试过程填写的问题报告和用户使用软件过程反馈的问题报告作为错误分类的信息。它包括 16 个类,在这 16 个类之下,还有 164 个子类。16 类有计算错误、逻辑错误、I/O 错误、数据加工错误、操作系

统和支持软件错误、配置错误、接口错误、用户需求改变(指用户在使用软件后提出软件无法满足的新要求所产生的错误)、预置数据库错误、全局变量错误、重复的错误、文档错误、需求实现错误(指软件偏离了需求说明产生的错误)、不明性质错误、人员操作错误、问题(指软件问题报告中提出的需要答复的问题)。

该分类方法特别适用于指导开发人员的缺陷消除和软件改进工作。通过对错误进行分类统计,可以了解错误分布状况,对错误集中的位置重点加以改进。该方法分类详细,适用面广,当然分类也较为复杂。该分类方法没有考虑造成缺陷的过程原因,不适用于软件过程改进活动。

4) IEEE 分类法

电气和电子工程师学会制定的软件异常分类标准(IEEE Standard Classification for Anomalies 1044—1993)对软件异常进行了全面的分类。该标准描述了软件生命周期各个阶段发现的软件异常的处理过程。分类过程由识别、调查、行动计划和实施处理四个步骤组成,每一步骤包括三项活动:记录、分类和确定影响。异常的描述数据称为支持数据项。分类编码由两个字母和三个数字组成。如果需要进一步的分类,可以添加小数。例如 RR 324、IV 321.1。RR 表示识别步骤,IV 表示调查步骤,AC 表示行动计划步骤,IM 表示确定影响活动,DP 表示实施处理步骤。分类过程的四个步骤都需要支持数据项。由于每个项目都有各自的支持数据项,该标准不强制规定支持数据项,但提供了各个步骤相关的建议支持数据项。强制分类建立通用的定义术语和概念,便于项目之间、商业环境之间、人员之间的交流沟通。可选分类提供对于特殊情况有用的额外的细节。在调查步骤,对实际原因、来源和类型进行了强制分类。其中调查步骤将异常类型分为逻辑问题、计算问题、接口/时序问题、数据处理问题、数据问题、文档问题、文档质量问题和强化问题(Enhancement),共八个大类,下面又分为数量不等的小类。分类细致深入,准确说明了异常的类型。

该分类方法提供一个统一的方法对软件和文档中发现的异常进行详细的分类,并提供异常的相关数据项帮助异常的识别和异常的跟踪活动。IEEE 软件异常分类标准具有较高的权威性,可针对实际的软件项目进行裁剪,灵活度高,应用面广。不足之处是没有考虑软件工程的过程缺陷,并且分类过程复杂。但是该方法提供了丰富的缺陷信息。缺陷原因分析活动可以充分利用这些信息进行原因分析。

5) 正交缺陷分类法

正交缺陷分类(Orthogonal Defects Classification,ODC)是 IBM 公司提出的缺陷分类方法。该分类方法提供一个从缺陷中提取关键信息的测量范例,用于评价软件开发过程,提出正确的过程改进方案。该分类方法用多个属性来描述缺陷特征。在 IBM ODC 最新版本里,缺陷特征包括八个属性:发现缺陷的活动、缺陷影响、缺陷引发事件、缺陷载体(Target)、缺陷年龄、缺陷来源、缺陷类型和缺陷限定词。ODC 对八个属性分别进行了分类。其中缺陷类型被分为八大类:赋值、检验(Checking)、算法、时序、接口、功能、文档、关联(Relationship)。由此看来,它的缺陷类型很简单,开发人员一般根据缺陷类型来修复程序,缺陷类型对于开发人员来说较容易理解,不会引起歧义。

该分类方法分类细致,适用于缺陷的定位、排除、缺陷原因分析和缺陷预防活动。缺陷特征提供的丰富信息为缺陷的消除、预防和软件过程的改进创造了条件。ODC 的缺点在于分类复杂,难以把握分类标准,缺陷分析人员的主观意见会影响属性的确定。

正交缺陷分类既是缺陷分类方法,同时又是软件缺陷度量分析方法,它介于统计缺陷模型和因果分析方法之间,在成本方面,ODC 和定量方法一样较低,在效果上却达到了定性分析的

力度。ODC 方法的缺陷数据为详细的过程分析奠定了基础，可以完整地分析全部缺陷的现象，做到对缺陷本质特性的分析和预防。

正由于正交缺陷分类的优异之处，ODC 自提出后，得到了广泛的发展与应用，全球多个软件组织都已经接受并使用 ODC。近几年，业界也开始研究并使用 ODC。作为定量的测量和分析方法，它已经成为 CMM4/5 的支撑工具之一，为 CMM4/5 定量过程管理、缺陷预防等提供有力支持。

2. 软件缺陷分类方法的应用

按照 CMM 和 GJB 5000—2003(CMM)的要求，参照正交缺陷分类 ODC，可在软件生命周期各个阶段中，根据相应阶段的评审和测试活动所提交的问题报告(包括各阶段中的问题建议报告)，确认缺陷的发现阶段和注入阶段，对缺陷进行分类。具体分类过程如下：根据评审人员或测试人员提交的缺陷报告(包括问题建议报告)重现缺陷，确认缺陷存在，并确认缺陷相关数据，一般包括发现缺陷的检测活动、引发缺陷的事件、缺陷来源、缺陷症状、缺陷的影响、缺陷类型、缺陷的严重性等，然后根据缺陷数据将缺陷进行分类，将缺陷划归某个产生该缺陷的软件过程。因此，实际应用中的缺陷分类通常是按照缺陷的表现形式、缺陷的严重程度、缺陷的优先级、缺陷的起源和来源、缺陷的根源以及缺陷的生命周期等。

1) 软件缺陷类型标准(即缺陷的表现形式)

(1) 10 F-Function(功能)。影响了重要的特性、用户界面、产品接口、硬件结构接口和全局数据结构，并且设计文档需要正式的变更；存在逻辑、指针、循环、递归、功能等缺陷。

(2) 20 A-Assignment(赋值)。需要修改少量代码，如初始化或控制块，以及声明、重复命名、范围、限定等。

(3) 30 I-Interface(接口)。与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表相互影响的缺陷。

(4) 40 C-Checking(检查)。提示的错误信息、不适当的数据验证等缺陷。

(5) 50 B-Build/package/merge(联编打包)。因配置库、变更管理或版本控制引起的错误。

(6) 60 D-Documentation(文档)。影响发布和维护，包括注释。

(7) 70 G-Algorithm(算法)。算法错误，如语法、标点符号、书写错误等。

(8) 80 U-User Interface(用户接口)。人机交互特性：屏幕格式、确认用户输入、功能有效性、页面排版等方面的缺陷。

(9) 90 P-Performance(性能)。不满足系统可测量的属性值，如执行时间、事务处理速率等。

(10) 100 N-Norms(标准)。不符合各种标准的要求，如编码标准、设计符号等。

2) 按缺陷的严重程度划分

按缺陷的严重程度划分，是指按软件的缺陷对软件质量的影响程度，即缺陷的存在对软件的功能和性能产生怎样的影响，按照严重程度由高到低的顺序可以分为 5 个等级。

(1) Critical：不能执行正常工作功能或重要功能，或者危及人身安全。

(2) Major：严重地影响系统要求或基本功能的实现，且没有办法更正(重新安装或重新启动该软件不属于更正办法)。

(3) Minor：严重地影响系统要求或基本功能错误地实现，但存在合理的更正办法(重新安装或重新启动该软件不属于更正办法)。

(4) Cosmetic：使操作者不方便或遇到麻烦，但它不影响执行工作或者重要功能。

(5) Other：其他错误。

需要说明的是，在具体的软件项目中，要根据实际情况来划分等级，不一定是5个等级。如果缺陷数目较少，则可以适当地减少等级。而一般的缺陷管理工具会自动地根据具体项目给出一个默认的缺陷严重程度。

同行评审错误的严重程度划分为Major(主要的、较大的缺陷)和Minor(次要的、小的缺陷)。

3) 按优先级划分

优先级指处理和修正软件缺陷的先后顺序的指标，即哪些缺陷需要优先修正，哪些缺陷可以稍后修正，按照优先级由高到低可以分为3个等级：high，middle，low。其中高优先级的缺陷是应该被立即解决的；中优先级的缺陷是指缺陷需要正常排队等待修复或列入软件发布清单；低优先级的缺陷是指缺陷可以在方便的时候被纠正。同缺陷的严重程度一样，优先级的划分也不是绝对的，可以根据具体的情况灵活划分。

例如，在项目开发期间原来标记为中的缺陷随着时间即将用尽，以及软件发布日期临近，可能变为低优先级。作为发现该软件缺陷的测试人员，需要继续监视缺陷的状态，确保自己能够同意对其所做的变动，并进一步提供测试数据或说服别人修正缺陷。

在这里需要说明的是，软件缺陷的严重程度和优先级是含义不同的但是又相互联系的两个概念，它们从不同的侧面描述了软件的缺陷对软件质量和最终用户的满意度的影响程度和处理的方式。

一般说来，严重程度高的缺陷通常具有较高的优先级。因为严重程度高的缺陷对软件质量的影响较大，应该优先处理，而严重程度低的缺陷可能只是软件不太完美，可以稍后再做处理。但是严重程度高的缺陷其优先级一定高吗？即缺陷的严重程度和缺陷的优先级一定成正比吗？答案是：不一定！例如：

(1) 严重程度高的优先级不一定高。软件本身是脆弱的，难以理清头绪，犹如一团乱麻，如果修复一个软件缺陷，需要重新修改软件的整体架构，由此可能造成牵一发而动全身——产生其他的缺陷，而且又有紧迫的产品发布等进度压力，修正软件将冒很大的风险，此时即使缺陷的严重程度很高，是否要修正，也仍需要做全面的考虑。

(2) 严重程度低的优先级不一定低。如果软件的界面不是很方便用户使用或软件的名字对公司的形象有一定的影响，这样的缺陷虽然不是很严重，但是它关系到软件和公司的市场形象，因而需要立即进行修正。

4) 按缺陷的起源和来源划分

软件缺陷的产生不仅仅是因为编程的错误，更多的是因为在软件开发的初期做了错误或不全面的需求分析和系统设计所引起的，因此根据产生缺陷的起源和来源可以划分成5类：Requirement，Architecture，Design，Code，Test。

缺陷起源指的是在需求、系统架构、设计、编码以及测试等不同阶段发现的缺陷。

缺陷来源指缺陷所在的地方，如文档、代码等。例如，Requirement指需求规格说明的错误，或需求说明书不清楚而引起的缺陷；Architecture指由于构架定义或设计以及接口定义或设计的问题引起的缺陷；Design指设计文档描述不准确和需求规格说明不一致引起的缺陷；Code指由于编码出错而引起的缺陷；Test指测试不彻底、不全面而遗留下的缺陷。

按照缺陷的来源对软件缺陷进行分类可以明确缺陷处理的负责人。

5) 按缺陷的根源划分

缺陷根源：指造成各种缺陷/错误的根本因素，以寻求软件开发流程的改进、管理水平的提高。具体如下。

(1) 测试策略：错误的测试范围，误解了测试目标，超越测试能力的目标等。

(2) 过程、工具和方法：无效的需求收集过程，过时的风险管理过程，不适用的项目管理方法，没有估算规程，无效的变更控制过程等。

(3) 团队/人：项目团队职责交叉，缺乏培训。没有经验的项目团队，缺乏士气和动机不纯等。

(4) 缺乏组织和通信：缺乏用户参与，职责不明确，管理失败等。

(5) 其他，如，硬件：处理器缺陷导致算术精度丢失，内存溢出等；软件：操作系统错误导致无法释放资源，工具软件的错误，编译器的错误，2000 年问题等；工作环境：组织机构调整，预算改变，罢工，噪音，中断，工作环境恶劣。

6) 按照缺陷的生命周期划分

Bug(缺陷)在英语中指的就是虫子，因此我们可以把缺陷看做有生命的小虫子，每一个缺陷都有一个从出生到死亡的周期，因此根据缺陷的生命周期可以这样划分：new，confirmed，fixed，closed，reopen 等。

每一个缺陷都是由测试人员发现并提交的，这个状态标注为 new(新建)；缺陷被提交后，由相应的负责人进行接受，即 comfirmed(确认)状态；相应的负责人员解决了该缺陷后，该缺陷的状态就改为 fixed(解决)，并且将其发给测试人员进行回归测试，防止产生其他错误；测试人员对已解决的缺陷进行回归测试，如果确定已经解决，那么缺陷的状态就改为 closed(关闭)，否则就需要返还给该缺陷的负责人重新修正；有的缺陷在以前的版本中已经关闭，但是在新的版本中又重新出现，则需要将其状态改为 reopen(重新打开)。

缺陷不同，其表现形式以及后果也不相同，在评审或测试过程中由于评审人员或测试人员的角度不同，对缺陷的认识也就不同，对缺陷的描述定义也就不会完全相同。如在设计评审阶段，关注的是软件设计是否满足需求分析的要求、软件功能是否被清晰描述；在单元测试阶段，采取“白盒”测试发现代码中的缺陷；在功能测试阶段，关注的是相对独立的功能模块或相关联的部件；在系统测试阶段，测试的则是软件产品的整体等。

在缺陷确认分类过程中可以分析不同阶段的缺陷情况，与标准缺陷类型进行关联，并能确认其注入阶段。软件缺陷类型标准与几个缺陷检测阶段存在缺陷及注入阶段关联情况如表 7-1 所示。

表 7-1 缺陷类型标准与软件测试阶段

缺陷类型标准		缺陷检测阶段				关联注入阶段
类型名称	描　述	设计评审	单元测试	功能测试	系统测试	
功能	功能实现，全局数据，与算法相对	需求满足性、数据库	功能、程序逻辑	程序错误	程序、数据库、遗漏需求	需求分析、设计
赋值	说明，重名，作用域，限制，初始化	函数说明	赋值	边界值	边界值、无效域	编码
接口/时序	过程调用和引用，函数调用，数据块共享，消息传递	过程接口	过程接口			设计
联编打包	变更管理，配置库，版本控制		打包问题	安装	集成(工具库、版本控制)	
文档	注释，需求、设计类文档	设计说明	设计问题	手册问题、需求问题、设计问题	手册及联机帮助	所有阶段

续表

缺陷类型标准		缺陷检测阶段				关联注入阶段
类型名称	描　　述	设计评审	单元测试	功能测试	系统测试	
用户接口/检查	人机交互，屏幕控制，出错信息、日志、输入输出	检查	检查	报表格式、界面控制、权限错误、提示信息	界面控制	设计、编码
算法	算法，局部数据结构，逻辑，指针，循环，递归	算法	算法			设计
标准/语法	程序设计规范，编码标准，指令格式等	规范	语法、规范			设计、编码
性能	不满足系统可测的属性值；执行时间、处理速率等				性能	设计
环境	设计，编译，测试，其他支持系统问题			测试环境	安装	集成、运行

缺陷分类无论是在软件开发、软件测试以及各个软件阶段的评审都得到了广泛的应用。缺陷属性针对文档和代码具有不同的实用性，表 7-2 列出了各自适用范围。

表 7-2　缺陷分类适用范围

缺 陷 属 性	软件测试	同行评审
缺陷标识(Identifier)	■	■
缺陷类型(Type)	■	■
缺陷严重程度(Severity)	■	■
解决优先级(Priority)	■	□
缺陷状态(Status)	■	□
缺陷起源(Origin)	■	■
缺陷原因(Cause)	●	●

■：需要记录　●：可以不考虑/可以记录　□：不考虑

7.1.4 软件缺陷管理流程

软件测试的任务是为了尽早发现软件系统中的缺陷，确保其修复，并最终保证软件的质量。缺陷跟踪管理是软件测试中的一个有机组成部分，是 CMM2 级的要求。在 CMM 第二级的软件组织中，软件项目从自身的需求出发，制定项目的缺陷管理流程。项目组将完整地记录开发过程中的缺陷，监控缺陷的修改过程，并验证修改缺陷的结果。

1. 缺陷管理的目标

软件缺陷能够引起软件运行时产生的一种不希望或不可接受的外部行为结果，软件测试过程简单说就是围绕缺陷进行的，对软件缺陷跟踪管理一般而言要达到以下目标。

(1) 确保每个被发现的缺陷都能够被解决。这里解决的意思不一定是被修正，也可能是其他处理方式(例如，在下一个版本中修正或不修正)。总之，对每个被发现的缺陷的处理方式必须能够在开发组织中达成一致。

(2) 收集缺陷数据并根据缺陷趋势曲线识别测试过程的阶段。决定测试过程是否结束有很多种方式，通过缺陷趋势曲线来确定测试过程是否结束是常用并且较为有效的一种方式。

(3) 收集缺陷数据并在其上进行数据分析,作为组织的过程财富。在对软件缺陷进行管理时,必须先对软件缺陷数据进行收集,然后才能了解这些缺陷,并且找出预防和修复它们的方法,以及预防引入新的缺陷。

2. 缺陷记录日志

缺陷记录日志(参见表 7-3)用于软件缺陷数据的收集,收集软件缺陷数据的步骤如下:

(1) 为测试和同行评审中发现的每一个缺陷做一个记录。

(2) 对每个缺陷要记录足够详细的信息,以便以后能更好地了解这个缺陷。

(3) 分析这些数据以找出哪些缺陷类型引起大部分的问题。

(4) 设计出能够发现和修复这些缺陷的方法(缺陷排除)。

表 7-3 缺陷记录日志

日期	编号	状态	类型	缺陷来源	排除来源	修改时间	修复时间
	描述						
	描述						

3. 软件缺陷管理流程

根据对国内外著名 IT 公司缺陷管理流程的研究,一般软件缺陷管理流程如图 7-3 所示。

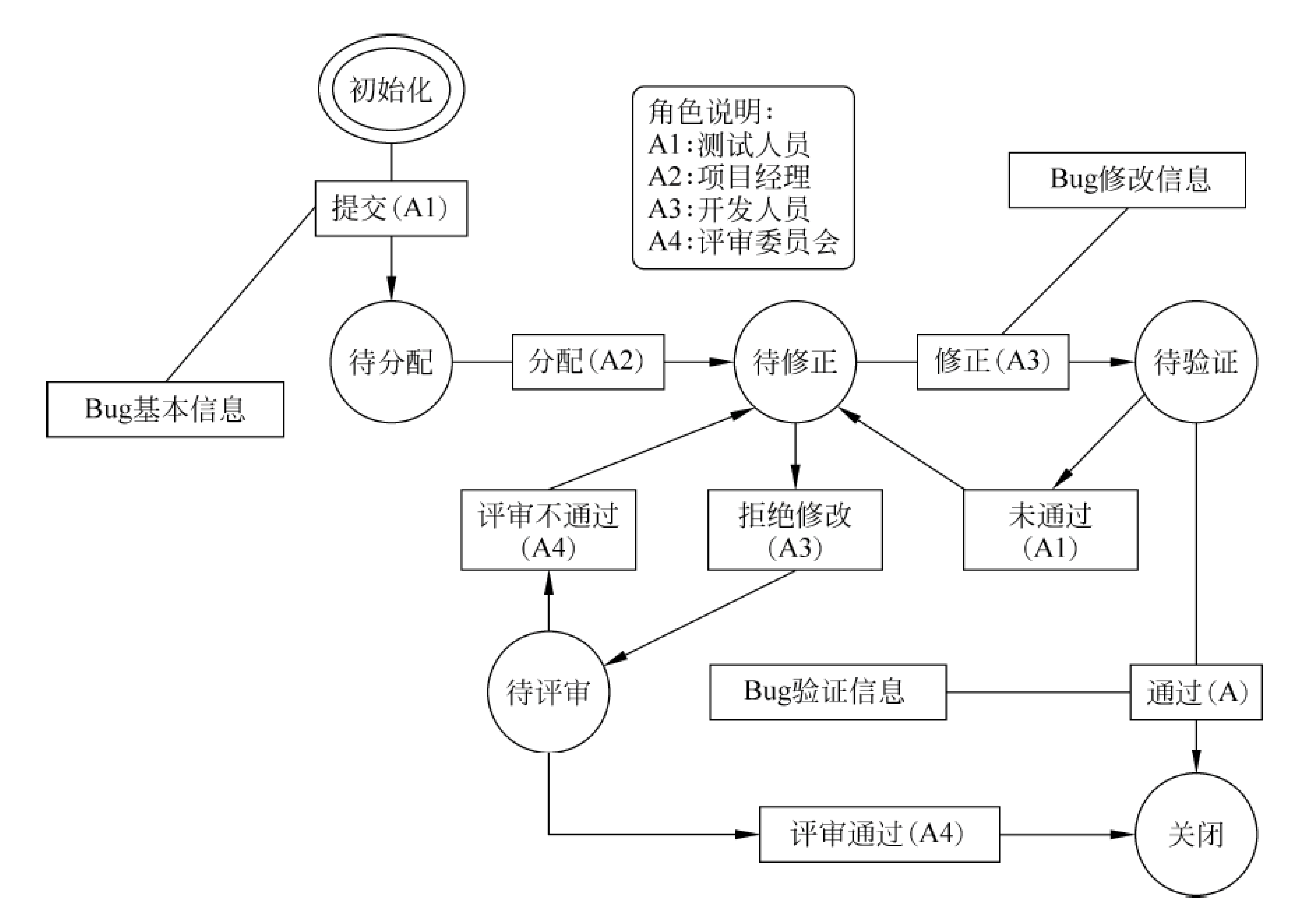


图 7-3 缺陷的一般管理流程

1) 缺陷管理流程中的角色

在缺陷管理流程中有 4 个角色:①测试人员 A1(进行测试的人员,并且是缺陷的发现者);②项目经理 A2(对整个项目负责,对产品质量负责的人员);③开发人员 A3(执行开发任务的人员,完成实际的设计和编码工作,以及对缺陷的修复工作);④评审委员会 A4(对缺陷进行最终确认,在项目成员对缺陷不能达成一致意见时,行使仲裁权力)。

2）缺陷管理流程中的缺陷状态

在缺陷管理流程中包含6种缺陷状态：①初始化（缺陷的初始状态）；②待分配（缺陷等待分配给相关开发人员处理）；③待修正（缺陷等待开发人员修正）；④待验证（开发人员已完成修正，等待测试人员验证）；⑤待评审（开发人员拒绝修改缺陷，需要评审委员会评审）；⑥关闭（缺陷已被处理完成）。

3）缺陷管理流程描述

软件缺陷管理流程描述如下：

（1）测试小组发现新的缺陷，并记录缺陷，此时缺陷状态为"初始化"。

（2）测试小组向项目经理提交新发现的缺陷（包括缺陷的基本信息），此时缺陷的状态为"待分配"。

（3）项目经理接收到缺陷报告后，根据缺陷的详细信息，确定处理方案，此时缺陷的状态为"待修正"。

（4）缺陷报告被分配给相应的开发人员，开发人员对缺陷进行修复，并填写缺陷的修改信息，然后等待测试人员对修复后的缺陷再一次进行验证，此时缺陷的状态为"待验证"。

（5）经测试人员验证后，发现缺陷未被修复，则重新交给原负责修复的开发人员，测试缺陷的状态为"待修正"。

（6）经测试人员验证后，认为缺陷被修复，则填写缺陷验证信息，缺陷修复完成，此时缺陷的状态为"关闭"。

（7）若测试人员验证缺陷未被修复，但是开发人员认为已修复完成拒绝再次修复，则将缺陷报告提交给评审委员会，等待评审委员会的评审，此时缺陷的状态为"待评审"。

（8）若评审委员会评审不通过，即软件缺陷未被修复，开发人员需继续修复，此时软件缺陷的状态为"待修正"。

（9）若评审委员会评审通过，即软件缺陷被修复，此时缺陷状态为"关闭"。

4）缺陷管理流程实施的注意事项

在整个缺陷的跟踪管理流程中，为了保证所发现的错误是真正的错误，需要有丰富测试经验的测试人员验证和确认发现的缺陷是否为真正的缺陷，发现的缺陷是由什么引起的，以及测试步骤是否准确、简洁、可以重复等。除此之外，由于对软件设计具体要求的不了解，对测试报告的个别软件错误，可能无法确认是否属于真正的软件错误，本地化服务商需要与软件供应商交流并确认；对于缺陷的处理都要保留处理信息，包括处理者姓名、时间、处理方法、处理步骤、错误状态、处理注释等；对缺陷的拒绝不能由程序员单方面决定，应该由项目经理、测试经理和设计经理组成的评审委员会决定；对于缺陷修复后必须由报告缺陷的测试人员验证后，确认已经修复，才能关闭缺陷。另外在缺陷跟踪管理流程中，还应注意以下几点：

（1）测试小组在提交事务时，应清楚详细地将问题描述出来，便于项目经理进行处理。

（2）项目经理在确定处理方案时，如对测试小组提出的事务有疑问，应及时与测试小组人员沟通，以保证完全理解测试小组提出的事务，确定正确的处理方案。同样，缺陷修复人员在处理事务时，如对测试小组提出的事务有疑问，也应及时与测试小组人员沟通，以保证准确处理测试小组提交的缺陷。

（3）修复人员在解决缺陷时，应将发现原因、解决的途径和方法详细地描述出来。以便日后的查阅。

（4）测试小组成员应定期整理和归类测试的Bug，并写成测试报告，向项目经理、技术总监报告测试结果。

7.2 软件缺陷度量、分析与统计

在软件开发过程中实施缺陷的度量与分析对于提高软件开发和测试效率，预防缺陷发生，保证软件产品质量有着十分重要的作用。另外，对软件的缺陷进行跟踪管理的目标之一是对缺陷的数据进行统计。通过对软件开发过程中发现的缺陷进行分析统计，可以判断软件质量、项目的进展。

7.2.1 软件缺陷度量

缺陷度量就是对项目过程中产生的缺陷数据进行采集和量化，将分散的缺陷数据统一管理，使其有序而清晰，然后通过采用一系列数学函数，对数据进行处理，分析缺陷密度和趋势等信息，从而提高产品质量和改进开发过程。缺陷度量是软件质量度量的重要组成部分，它和软件测试密切相关。尽管缺陷度量本身并不能发现缺陷、剔除缺陷，但是有助于这些问题的解决。

软件缺陷度量的方法较多，从简单的缺陷计数到严格的统计建模，其主要的度量方法有缺陷密度（软件缺陷在规模上的分布）、缺陷率（缺陷在时间上的分布）、整体缺陷清除率、阶段性缺陷清除率、缺陷趋势、预期缺陷发现率等。

1. 缺陷密度

Myers 有一个关于软件测试的著名的反直觉原则：在测试中发现缺陷多的地方，还有更多的缺陷将会被发现。他认为：缺陷发现多的地方漏掉的缺陷也可能会多，或者说在测试效率没有被显著改善之前，在纠正缺陷时可能会引入较多的错误。这条原理的数学表述就是缺陷密度的度量——每千行代码或每个功能点（或类似功能点的度量——对象点、数据点、特征点等）的缺陷数，缺陷密度越低意味着产品质量越高。

如果缺陷密度跟上一个版本相同或更低，就应该分析当前版本的测试效率是不是降低了？如果不是，意味着质量的前景是乐观的；如果是，那么就需要额外的测试，还需要对开发和测试的过程进行改善。

如果缺陷密度比上一个版本高，那么就应该考虑在此之前为显著提高测试效率所进行的有关策划是否需要在本次测试中实施？如果是，虽然需要开发人员更努力去修正缺陷，但对质量的保证是有益的；如果不是，则意味着质量恶化、质量很难得到保证。这时，要保证质量，就必须延长开发周期或投入更多的资源。

对于一个软件工作产品，软件缺陷分为两种：通过评审或测试等方法发现的已知缺陷（检测缺陷）、尚未发现的潜伏缺陷（残留缺陷）。缺陷密度的定义如下：

$$\text{缺陷密度}=\frac{\text{检测缺陷的数量}}{\text{产品规模}}$$

在缺陷密度公式中，产品规模的度量单位可以是文档页、代码行、功能点。缺陷密度是软件缺陷的基本度量，可用于设定产品质量目标，支持软件可靠性模型（如 Rayleigh 模型），预测残留缺陷，进而对软件质量进行跟踪和管理。缺陷密度还支持基于缺陷统计的软件可靠性增长模型（如 Musa-Okumoto 模型）对软件质量目标进行跟踪并评判能否结束软件测试。

2. 缺陷率

缺陷率的通用概念是一定时间范围内的缺陷数与出现错误的概率（Opportunities For

Error,OFE)的比值。用它可近似估算软件中的缺陷数目。

软件产品缺陷率,即使对一个特定的产品,在其发布后不同时段也是不同的。例如,从应用软件的角度来说,90%以上的缺陷是在发布后两年内被发现的,而对操作系统来说,90%以上的缺陷通常在产品发布后需要四年的时间才能被发现。

3. 缺陷清除率

缺陷清除率(也称"缺陷排除率"),英文缩写为DER(Defect Elimination Rate)。它可以用做缺陷的预测和分析。

DER分为两种:整体缺陷清除率和阶段缺陷清除率,阶段性的缺陷清除率能够反映整体缺陷清除率。

$$缺陷清除率=\frac{检测缺陷}{所有缺陷}$$

$$由于所有缺陷不容易确定,故缺陷清除率\approx\frac{检测缺陷}{(检测缺陷+以后发现的缺陷数)}$$

1) 整体缺陷清除率

先引入几个变量,F 为描述软件规模用的功能点;D_1 为在软件开发过程中的检测缺陷数;D_2 为软件发布后的检测缺陷数;D 为检测的总缺陷数。因此,$D=D_1+D_2$。

对于一个应用软件项目,则有以下计算方程式(从不同的角度估算软件的质量):

质量 $= D_2/F$;

缺陷注入率 $= D/F$;

整体缺陷清除率$= D_1/D$;

假如有100个功能点,即 $F=100$,而在开发过程中发现了20个错误,提交后又发现了3个错误,则 $D_1=20$,$D_2=3$,$D=D_1+D_2=23$

质量(每功能点的缺陷数)$=D_2/F= 3/100=0.03$ (3%)

缺陷注入率$=D/F=20/100=0.20$ (20%)

整体缺陷清除率$=D_1/D=20/23=0.8696$ (86.96%)

有资料统计,美国的平均整体缺陷清除率目前只达到大约85%,而对一些具有良好的管理和流程的著名软件公司,其主流软件产品的缺陷清除率可以超过98%。

众所周知,清除软件缺陷的难易程度在各个阶段也是不同。需求错误、规格说明与设计问题及错误修改是最难清除的,如表7-4所示。

表7-4 不同缺陷源的清除效率

缺陷源	总缺陷数	清除效率/%	残留缺陷数
需求报告	1.00	77	0.23
设计	1.25	85	0.19
编码	1.75	95	0.09
文档	0.60	80	0.12
错误修改	0.40	70	0.12
合计	5.00	85	0.75

表7-5反映的是CMM五个等级是如何影响软件质量的,其数据来源于美国空军1994年委托SPR(美国一家著名的调查公司)进行的一项研究。从表7-5中可以看出,CMM级别越高,缺陷清除率也越高。

表 7-5　SEI CMM 级别潜在缺陷与清除率

SEI CMM 级别	总缺陷数	清除效率/%	残留缺陷数
1	5.00	85	0.75
2	4.00	89	0.44
3	3.00	91	0.27
4	2.00	93	0.14
5	1.00	95	0.05

2）阶段性缺陷清除率

阶段性缺陷清除率是缺陷密度度量的扩展。除测试外，它要求跟踪开发周期所有阶段中的缺陷，包括需求评审、设计评审、代码审查。因为编程中的大部分缺陷是与设计问题有关联的，进行正式评审或功能验证以增强前期过程的缺陷清除率有助于减少缺陷的注入。阶段的缺陷清除模型在某种程度上能够反映开发工程总的缺陷清除能力。

下面是缺陷清除有效性(Defect Remove Efficiency，DRE)的分析，DRE 可以定义为：

$$\text{DRE}=\frac{\text{开发阶段清除的缺陷数}}{\text{产品缺陷总数}}\times 100\%$$

因为产品缺陷的总数是不知道的，必须通过一些方法获得其近似值，如经典的种子公式方法，或近似等于当前阶段清除的缺陷数＋以后发现的缺陷数。当 DRE 用于前期的和特定阶段的时候，此时 DRE 相应地被称为早期缺陷清除有效性和阶段有效性，对给定阶段的残留缺陷数，可以估计为：

$$\text{当前阶段的残留缺陷数}=\frac{\text{本阶段检测缺陷数}}{\text{本阶段入口残留的缺陷数}+\text{本阶段注入的缺陷数}}\times 100\%$$

给定阶段的 DRE 度量值越高，遗漏到下一个阶段的缺陷就越少。

缺陷是在各个阶段注入到阶段性产品或者成果中去，通过表 7-6 描述的与缺陷注入和清除相关联的活动分析，可以更好地理解缺陷清除有效性。回归缺陷是由于修正当前缺陷时而引起相关的、新的缺陷，所以即使在测试阶段，也会产生新的缺陷。

表 7-6　与缺陷注入和清除相关联的活动

开 发 阶 段	缺 陷 注 入	缺 陷 清 除
需求	需求收集过程和功能规格说明书	需求分析和评审
系统/概要设计	设计工作	设计评审
详细/程序设计	设计工作	设计评审
编码和单元测试	编码	代码审查、测试
集成测试	集成过程、回归缺陷	构建验证、测试
系统测试	回归缺陷	测试、评审
验收测试	回归缺陷	测试、评审

清除的缺陷数等于检测到的缺陷数减去不正确修正的缺陷数＋未修正的缺陷数。如果不正确修正的缺陷数＋未修正的缺陷数所占的比例很低(经验数据表明，测试阶段大概为 2%)，清除的缺陷数就近似于检测缺陷数。

4. 缺陷趋势

“趋势”一词在词典里的解释是“事物发展的动向”，也就是会呈现出某种规律，可用来对未来进行预测。缺陷趋势是在一定周期时间或者一定阶段内，产生/发现缺陷的动向或规律，它是缺陷率按时间或者按阶段增长/下降的一个动态分布。周期可以是天、周、月。阶段可以是

版本，如 1.2，1.3，1.3。通常缺陷趋势用缺陷趋势图来表示。

5. 预期缺陷发现率

缺陷发现率，英文缩写为 DDR(Defect Discovery Rate)，描述在特定时间段内发现缺陷数目的一种度量，常常以图表形式来显示。计算方法是计算测试人员各自发现的缺陷数总和除以各自所花费的测试时间总和。

$$\text{缺陷发现率} = \frac{\sum \text{提交缺陷数(个)}}{\sum \text{执行测试的有效时间(小时)}}$$

预期缺陷发现率则是通过对缺陷发现率的分析，预期在将来的某段时间内可能发现的缺陷数目。

$$\text{预期缺陷发现率} = \frac{\sum \text{可能发现的缺陷数(个)}}{\sum \text{未来的某段时间内(小时)}}$$

许多组织都把缺陷发现率当做一个帮助自己判断测试是否可以结束、预测产品发布日期的重要度量。如果缺陷发现率降到规定水平以下，通常都会做好产品发布的准备。

虽然缺陷发现率趋势下降一般都是一个不错的信息，但是，我们必须提防其他可能导致发现率下降的因素（工作量减少、没有新的测试用例，等等）。所以我们的重要决策往往要依据多个支撑性度量元。

正常来讲，到测试后期，每天发现的新缺陷的数量呈下降趋势，如果我们假定每天工作量恒衡定的，那么每发现一个缺陷所消耗的成本也会呈现上升趋势。到某个点后，继续进行测试发现缺陷所需成本已经超过缺陷本身带来的损失。这时候测试可以退出了。

7.2.2 软件缺陷分析

缺陷分析是将软件开发各个阶段产生的缺陷信息进行分类和汇总统计，计算分析指标，编写分析报告的活动。

1. 缺陷分析的意义

通过软件缺陷分析可以发现各种类型缺陷发生的概率，掌握缺陷集中的区域、明确缺陷发展趋势、挖掘缺陷产生的根本原因，便于有针对性地提出遏制缺陷发生的措施、降低缺陷数量。缺陷分析报告中的统计数据及分析指标既是对当前软件质量状况的评估，也是判定软件是否能按期发布或交付使用的重要依据。实施缺陷分析的前提是需要一个符合项目要求的缺陷数据管理系统，通过采集完整的缺陷数据信息，进行缺陷数据分析，达到改进软件过程质量并实施缺陷预防措施的目的。

缺陷分析也可以用来评估当前软件的可靠性，并且预测软件产品的可靠性变化，缺陷分析在软件可靠性评估中占有很重要的地位。

另外，通过缺陷分析达到缺陷预防的目的，这是缺陷管理的核心任务之一。

2. 缺陷预防

缺陷预防的着眼点在于寻找缺陷的共性原因。通过寻找、分析和处理缺陷的共性原因，实现缺陷预防。缺陷预防并不是一个不切实际的目标，测试人员在开发过程中应该积极为开发小组提供缺陷分析，就有可能降低缺陷产生的数量。因此，缺陷管理的最终目标是预防缺陷，不断提高整个开发团队的技能和实践经验，而不只是修正它们。

缺陷预防策略非常简单和容易实现，策略是①测试活动尽量提前，通过及时消除开发前期阶段引入的缺陷，防止这些缺陷遗留并放大到后续环节；②通过对已有缺陷进行分析，找出产

生这些缺陷的技术上的不足和流程上的不足(缺陷的根源),然后寻找一个方法来对这些不足进行改进,预防类似的缺陷在将来出现。这种策略费用并不高,但能带来极大的好处。

3. 缺陷分析步骤

缺陷分析的第一步是记录缺陷,值得注意的是记录缺陷不应该满足于记录缺陷的表面症状。测试的一个重要职责就是试图发现缺陷的根本原因,在测试时不应将产品看做一个黑盒,而应该像开发人员那样了解产品的内在特性,包括深入源代码,理解产品的设计和实现。

缺陷分析的第二步是对测试出来的缺陷进行缺陷分类,找出那些关键的缺陷类型,进一步分析其产生的根源,针对性地制定改进措施。缺陷分析非常关键的一步就是寻找一个预防类似缺陷再次发生的方法。这一方法不仅涉及开发人员、测试人员,还涉及不直接负责代码编写的资深技术人员。利用这一阶段的实践成果,开发人员可以预防缺陷的发生,而不仅仅是修正这些缺陷。

缺陷分析的第三步是进行缺陷预防分析,它是整个缺陷分析过程的核心。这一阶段总结出的实践可以在更广泛的范围内预防潜在的缺陷。由于分析结果的广泛应用性,分析某个具体缺陷的投入将很容易被收回。在这个时候,缺陷分析提供了两个非常重要的参数,一个是缺陷数量的趋势,另一个是缺陷修复的趋势。缺陷趋势就是将每月新生成的缺陷数、每月被解决的缺陷数和每月遗留的缺陷数绘制成一个趋势图表。

一般在项目的开始阶段发现缺陷数曲线会呈上升趋势,到项目中后期被修复缺陷数曲线会趋于上升,而发现缺陷数曲线应总体趋于下降。同时处于 Open 状态的缺陷也应该总体呈下降趋势,到项目最后,三条曲线都趋向于零。项目经理可通过持续观察这张图表,确保项目开发健康发展。同时,通过分析预测项目测试缺陷趋于零的时间,以制定产品质量验收和发布的时间。

缺陷分析的最后一步是编写缺陷分析报告,绘制缺陷分析图。

缺陷分析报告中的统计数据及分析指标既是对软件质量的权威评估,也是确定测试是否达到结束标准、判定测试是否已达到客户可接受状态和判定软件是否能发布或交付使用的重要依据。

另外,缺陷分析图表会告诉我们很多有价值的信息。例如,可分析开发和测试在人力资源的配比上是否恰当,可以分析出某个严重的缺陷所造成的项目质量的波动。对于异常的波动,如本来应该是缺陷曲线随着测试工作的持续开展最后是收敛的,却到了某个点发现的故障数反而呈上升趋势,那么意味着往往有一些特殊事件的发生。通过对测试缺陷进行分析,能够给予我们很多改进研发和测试工作的信息。

4. 缺陷分析方法

在缺陷分析中,常用的主要缺陷参数有四个,即①状态:缺陷的当前状态(打开的、正在修复或关闭的等);②优先级:必须处理和解决缺陷的相对重要性;③严重性:缺陷的相关影响,对最终用户、组织或第三方的影响等;④起源:导致缺陷的起源故障及其位置,或排除该缺陷需要修复的构件。

可以将缺陷计数作为时间的函数来报告,即创建缺陷趋势图或报告;也可以将缺陷计数作为一个或多个缺陷参数的函数来报告,如作为缺陷密度报告中采用的严重性或状态参数的函数。这些分析类型分别为揭示软件可靠性的缺陷趋势或缺陷分布提供了判断依据。

例如,预期缺陷发现率将随着测试进度和修复进度而最终减少。可以设定一个阈值,在缺陷发现率低于该阈值时才能部署软件。也可根据执行模型中的起源报告缺陷计数,以允许检测“较差的模块”、“热点”或需要再三修复的软件部分,从而指示一些更基本的设计缺陷。

这种分析中包含的缺陷必须是已确认的缺陷。不是所有的实际缺陷都必须报告,这是因

为某些缺陷可能是扩展请求，超出了项目的规模，或描述的是已报告的缺陷。然而，需要查看并分析一下，为什么许多报告的缺陷不是重复的缺陷就是未经确认的缺陷，这样做是有价值的。

国内外进行缺陷分析常用以下几种方法。

(1) ODC 缺陷分析：由 IBM 的 Waston 中心推出。Phontol.com 将一个缺陷在生命周期的各环节的属性组织起来，从单维度、多维度来对缺陷进行分析，从不同角度得到各类缺陷的缺陷密度和缺陷比率，从而积累得到各类缺陷的基线值，用于评估测试活动、指导测试改进和整个研发流程的改进；同时根据各阶段缺陷分布得到缺陷清除过程特征模型，用于对测试活动进行评估和预测。Phontol.com 所涉及的缺陷分布、缺陷趋势等都属于这个方法中的一个角度而已。

(2) Gompertz 分析：根据测试的累积投入时间和累积缺陷增长情况，拟合得到符合自己过程能力的缺陷增长 Gompertz 曲线，用来评估软件测试的充分性、预测软件极限缺陷数和退出测试所需时间、作为测试退出的判断依据、指导测试计划和策略的调整。

(3) Rayleigh 分析：通过生命周期各阶段缺陷发现情况得到缺陷 Rayleigh 曲线，用于评估和预测软件质量。

(4) 四象限分析：根据软件内部各模块、子系统、特性测试所累积时间和缺陷清除情况，与累积时间和缺陷清除情况的基线进行比较，得到各个模块、子系统、特性测试分别位于的区间，从而判断哪些部分测试可以退出、哪些测试还需加强，用于指导测试计划和策略的调整。

(5) 根本原因分析：利用鱼骨图、柏拉图等分析缺陷产生的根本原因，根据这些根本原因采取措施，改进开发和测试过程。

(6) 缺陷注入分析：对被测软件注入一些缺陷，通过已有用例进行测试，根据这些刻意注入缺陷的发现情况，判断测试的有效性、充分性，预测软件残留缺陷数。

(7) DRE/DRM 分析：通过已有项目历史数据，得到软件生命周期各阶段缺陷注入和排除的模型，用于设定各阶段质量目标，评估测试活动。

7.2.3 软件缺陷统计

软件缺陷统计是软件分析报告中的重要内容之一。事实上，从统计的角度出发，可以对软件过程的缺陷进行度量，如软件功能模块缺陷分布、缺陷严重程度分布、缺陷类型分布、缺陷率分布、缺陷密度分布、缺陷趋势分布、缺陷注入率/消除率等。统计的方式可以用表格，也可用图表表示，如散点图、趋势图、因果图、直方图、条形图、排列图等。

1. 软件功能模块的缺陷统计

图 7-4 中的软件包含文件、编辑、视图、插入、格式、工具、帮助等功能模块。该图说明的是，在某些模块，执行的测试用例多，但是没有成比例地发现很多缺陷，所以这些模块是比较成熟的，因为这些模块几乎不怎么修改，再测试的话，也不会发现什么问题的；但是某些模块执行的测试用例少，却发现了比较多的缺陷，这些模块修复的地方，或者发生功能变更的可能性大，所以将成为质量不稳定的关键点。并且在以后的回归测试中，应该在质量不稳定的模块中投入更多的人手和时间，进行更全面的测试，其他模块就相应减少测试工作的投入。这样，测试工作的压力就不是那么大了，而且效率也会相对地提高。

2. 缺陷严重程度统计

按照缺陷严重程度及阶段缺陷分布可以统计整个项目生命周期中所有同行评审的缺陷分布，也可以统计某一阶段所有同行评审的缺陷分布，如图 7-5 所示。

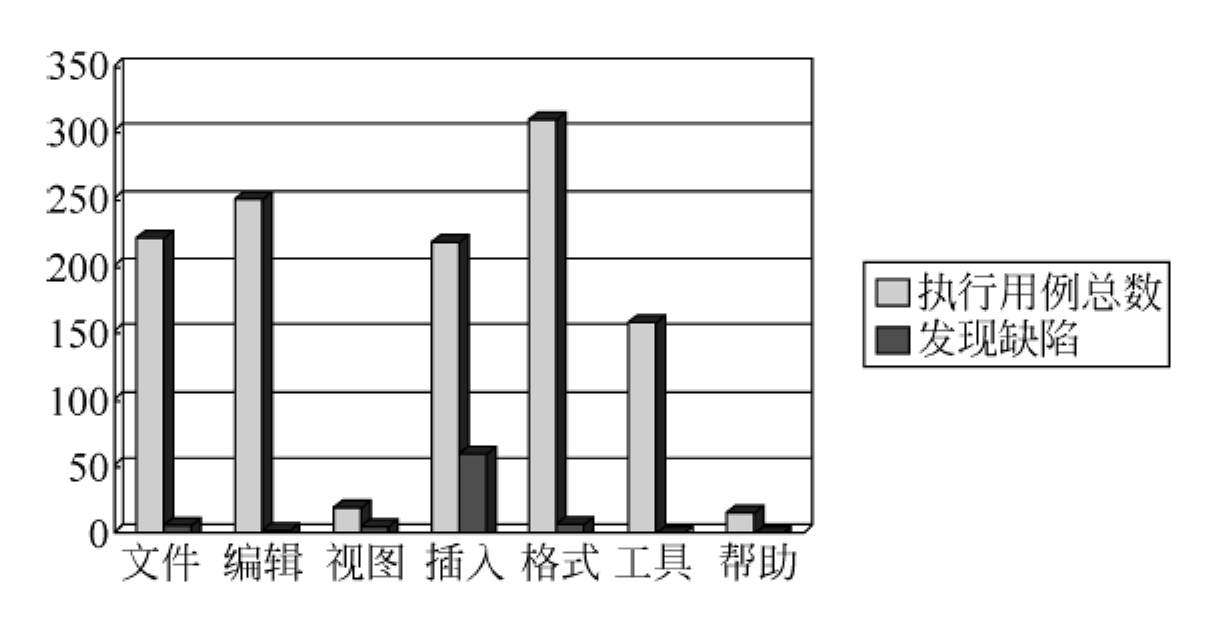


图 7-4 软件功能模块缺陷分布图

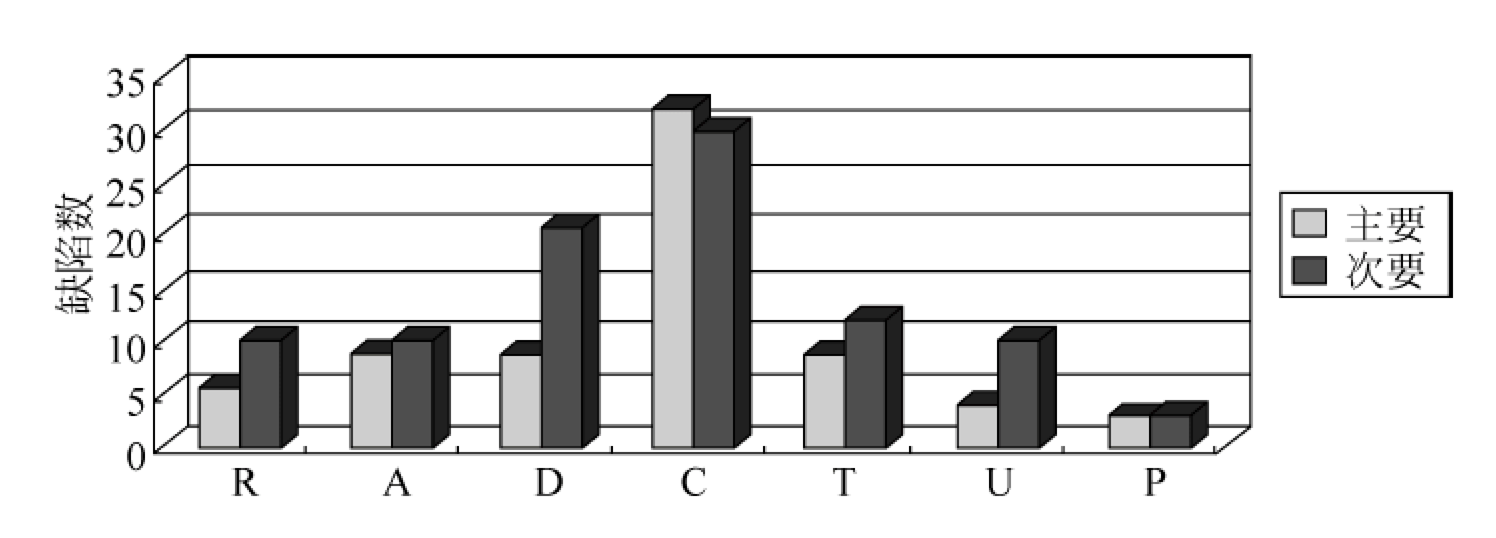


图 7-5 缺陷严重程度按阶段缺陷分布的分布图

多个项目按阶段缺陷率的统计分布图如图 7-6 所示。

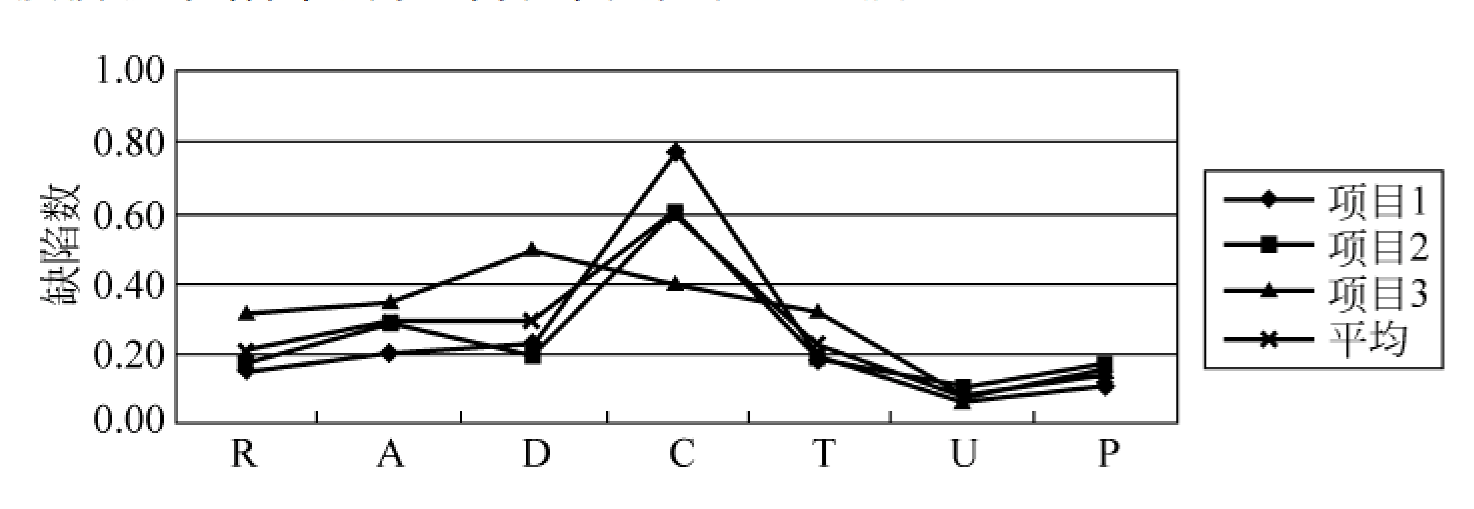


图 7-6 多个项目按阶段缺陷率的统计分布图

3. 缺陷类型统计

按照缺陷类型统计分布图,可以是某一次评审的缺陷统计,也可以是某一类型缺陷评审的缺陷统计,也可以是某一阶段所有同行评审的缺陷统计,也可以是整个项目周期内所有同行评审的缺陷统计,如图 7-7 所示。

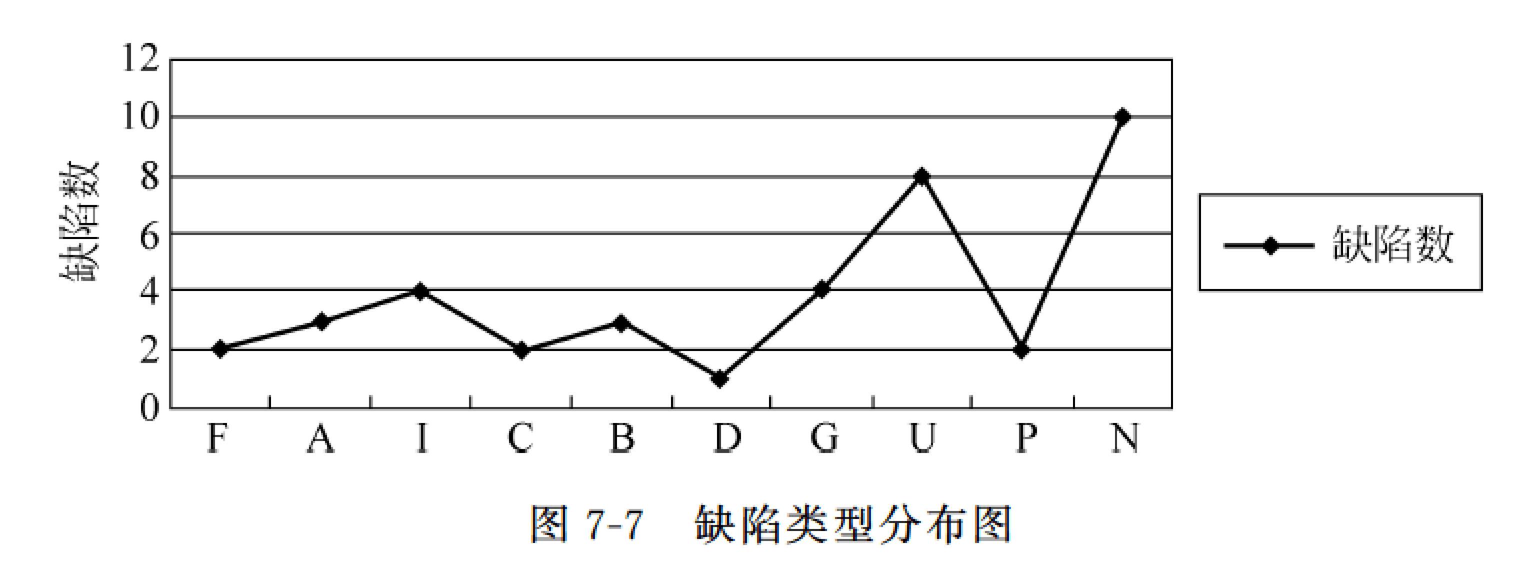


图 7-7 缺陷类型分布图

我们可以根据图 7-7 找出那些关键的缺陷类型,进一步分析其产生的根源,从而针对性地制定改进措施。图 7-8 所示为某软件系统测试的缺陷类型饼图。

从系统测试故障来看,有较多故障是由接口原因造成的,细分有以下几种原因。

(1) 跨项目间的接口,接口设计文档的更改没有建立互相通知的机制,导致接口问题到系

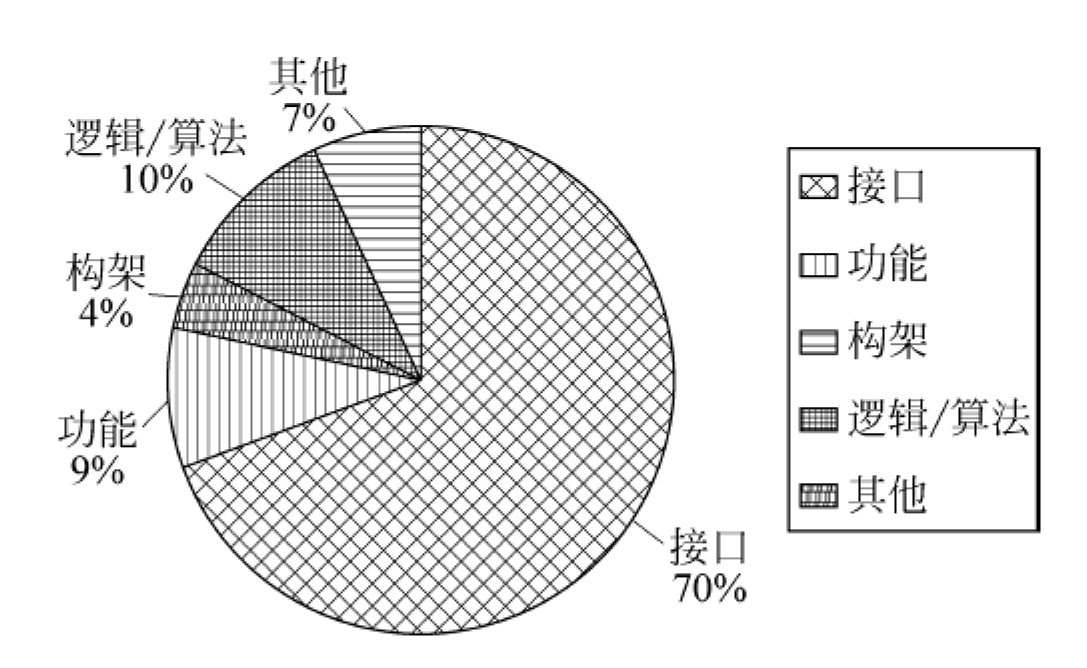


图 7-8 系统测试缺陷类型分布图

统测试时才暴露。

(2) 部门内部跨子系统的接口,由于本项目设计文档是按功能规划编写的,而不是按照产品组件,一般由主要承接功能工作的组编写该文档,接口内容可能不为其他开发组理解并熟悉,导致因接口问题而出错。

(3) 系统设计基线化后,更改系统接口,没有走严格的变更流程,进行波及影响分析,导致该接口变更只在某个子系统中被修改,而使错误遗漏下来。

为此,我们可以针对性地制定改进建议:

(1) 对接口文档的评审一定要识别受影响的相关干系人,使他们了解并参与接口设计的把关。

(2) 对基线化的接口设计文档的变更一定要提交变更单给变更控制委员会(CCB)决策,并做好充分的波及影响分析,以便同步修改所有关联的下游代码。

(3) 概要设计文档按子系统规划,详细设计文档按模块规划,通知相关组参加评审协调接口设计。

以上例子的缺陷分类只是为了描述方便,本身描述并不尽合理。实际定义缺陷分类可能有很多个维度,如发现活动、引入活动、缺陷来源、缺陷类型、严重程度等。只要满足自己的缺陷管理、缺陷分析需求即可。

4. 缺陷趋势图

项目管理中一项非常重要但也十分困难的工作是衡量项目的进度、质量、成本等,统称为项目的状态,以确定项目是否能按期保质完成。这方面,测试提供了两个非常重要的参数,一个是缺陷数量的趋势(如图 7-9 所示),另一个是缺陷修复的趋势(这里所说的测试均指系统测试)。

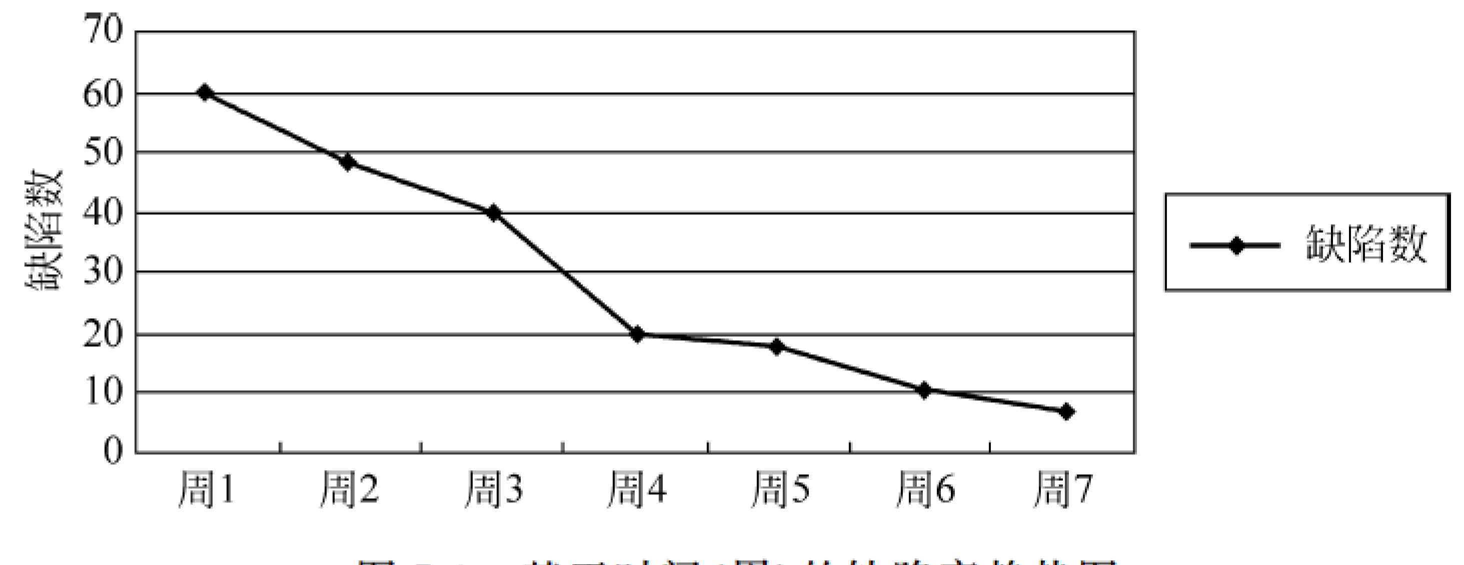


图 7-9 基于时间(周)的缺陷率趋势图

缺陷趋势就是将每月/每周新生成的缺陷数、每月/每周被解决的缺陷数和每月/每周遗留的缺陷数标成一个趋势图表。一般在项目的开始阶段发现缺陷数曲线会呈上升趋势,到项目

中后期被修复缺陷数曲线会趋于上升；而发现缺陷数曲线应总体趋于下降；同时处于 Open 状态的缺陷也应该总体呈下降趋势，到项目最后，三条曲线都趋向于零，如图 7-10 所示。

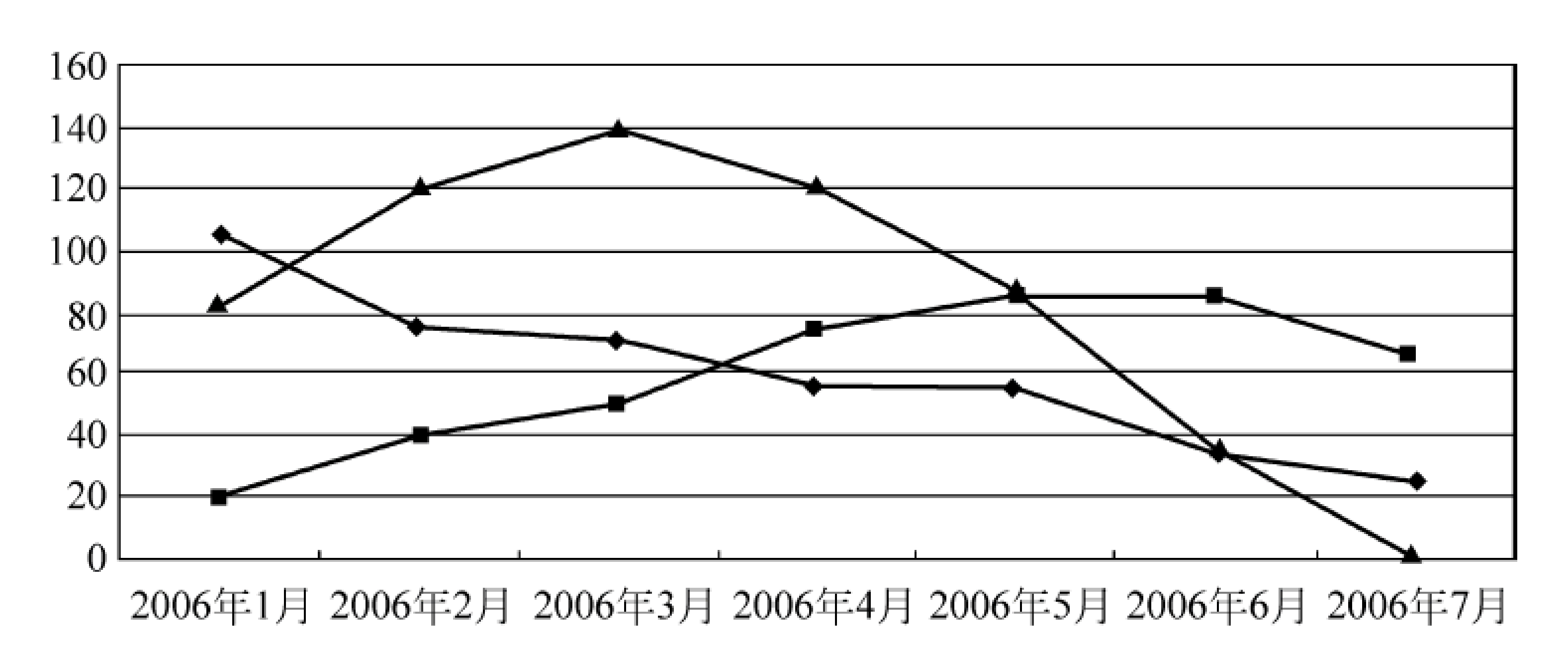


图 7-10 缺陷发现、修复、收敛趋势图

项目管理者会持续观察这张图表，确保项目健康发展，同时通过分析预测项目测试缺陷趋于零的时间。在一定的历史经验的基础上分析使用这一图表会得到很多有价值的信息，比如说，可分析开发和测试在人力资源的配比上是否恰当，可以分析出某个严重的缺陷所造成的项目质量的波动。对于异常的波动，如本来应该越测试越收敛的，却到了某个点，发现的故障数反而呈上升趋势，那么，这些点往往有一些特殊事件发生。例如，在该时间段要进行回归测试的版本增加了新的功能，导致缺陷引入；该回归版本没有进行集成测试就直接进行最终的系统测试等。

当然，这个统计周期也可以根据我们的项目实施情况进行。如按照回归版本的版本号进行统计、按周进行统计等。也有公司把缺陷收敛情况当做判断版本是否可以最终外发的一个标志。

项目的缺陷率按版本的趋势见图 7-11。

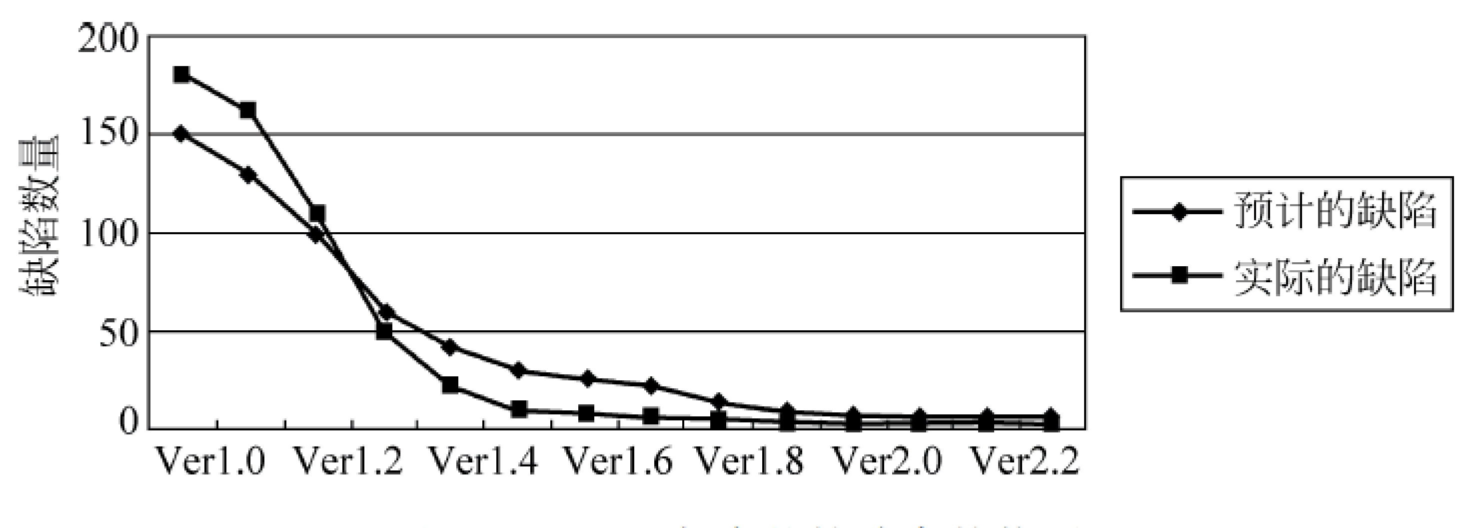


图 7-11 基于版本的缺陷率趋势图

这个缺陷趋势分析图，说明了软件在测试版本的 Ver 1.4 的时候，软件的质量已经得到了很好的控制了，在 Ver 1.8 的时候，基本上就已经可以发布软件了，后面的测试几乎是没有什么意义的。原因很简单，软件中的缺陷既然是不可能全部发现的，就不要指望找出软件中全部的缺陷，当它足够少(各公司的定义是不同的)的时候，就应该停止测试了。

7.3 软件缺陷报告

一旦缺陷被发现后，测试人员下一步要做的工作就是要就所发现的缺陷与开发人员进行沟通，最简单的沟通方法就是通过缺陷报告。缺陷报告的重要性不仅仅在于文档化，同时也是使发现的缺陷被修正的唯一方法；另外，缺陷报告记录了缺陷发生的环境，如各种资源的配置

情况、缺陷的再现步骤以及缺陷性质的说明，更重要的是，它还记录着缺陷的处理过程和状态。而缺陷的处理过程从一定程度反映了测试的进程和被测软件的质量状况以及改善过程；最后缺陷报告也是测试人员测试工作的可见产品，如果缺陷报告不能做到清晰、完整和易理解，并且缺陷不能重现，则缺陷被拒绝修改或者开发人员无法重现缺陷的可能性就会大大增加。

在软件测试过程中，每发现一个软件错误（缺陷），都要记录该错误的特征和复现步骤等信息，以便相关人士分析和处理软件错误。为了便于管理测试发现的软件错误，通常要采用软件缺陷数据库，并将每一个发现的错误输入到软件缺陷数据库中。软件缺陷数据库的每一条记录称为一个软件缺陷报告。

提供准确、完整、简洁、一致的缺陷报告是体现软件开发、测试与管理的专业性、高质量的主要评价指标。每个软件问题报告只书写一个缺陷或错误，这样可以每次只处理一个确定的错误，定位明确，提高效率，也便于错误修复后的验证。

在缺陷的生命周期（缺陷从开始提出到最后完全解决，并通过复查的过程）中，缺陷报告的状态不断变化着，它记录着缺陷的处理进程。

7.3.1 缺陷报告内容

不同的项目和测试机构会依据不同的标准和规范来编制缺陷报告，目的是为缺陷报告阅读者识别缺陷提供足够的信息。

1. 缺陷报告主要内容

一般情况下，缺陷报告主要包含下列内容。

（1）问题报告编号：为了便于对缺陷的管理，每个缺陷必须赋予一个唯一编号，编号规则可根据需要和管理要求制定。

（2）标题：标题用简明的方式传达缺陷的基本信息，标题应该简短并尽量做到唯一，以便在观察缺陷列表时，可以很容易地注意到。

（3）报告人：缺陷报告的原始作者，有时也可以包括缺陷报告的修订者。当负责修复该缺陷的开发人员对报告有任何异议/疑义时，可以与报告人联系。

（4）报告日期：首次报告的日期。让开发人员知道创建缺陷报告的日期是很重要的，因为有可能这个缺陷在前边版本曾经修改过。

（5）程序（或者组件）的名称：可分辨的被测试对象。

（6）版本号：测试可能跨越多个软件版本，提供版本信息可以方便进行缺陷管理。

（7）配置：发现缺陷的软件和硬件的配置。如操作系统的类型、是否有浏览器载入、处理器的类型和速度、RAM 的大小、可用的 RAM、正在运行的其他程序，等等。

（8）缺陷的类型：如代码错误、设计问题、文档不匹配等。

（9）严重性：描述所报告缺陷的严重性。

（10）优先级：由开发人员或管理人员进行确定，依据修复这个缺陷的重要性而定。

（11）关键词：以便分类查找缺陷报告，关键词可在任何时候添加。

（12）缺陷描述：对发现的问题进行详细说明，尽管描述要深入，但是简明仍是最重要的。缺陷描述的主要目的是说服开发人员决定去修复这个缺陷。

（13）重现步骤：这些步骤必须是有限的，并且描述的信息足够读者知道正确地执行就可以重现这个缺陷。

（14）结果对比：在执行了重现缺陷步骤后，期望发生什么，实际上又发生了什么。

2. 报告缺陷的基本原则

在软件测试过程中，对于发现的大多数软件缺陷，要求测试人员简捷、清晰地把发现的问

题报告提交给需要判断是否进行修复的小组，使其得到所需要的全部信息，然后才能决定怎么做。缺陷报告是测试人员的主要工作产品之一，好的缺陷报告会增加开发人员对测试人员的信任度，提高开发效率。因此，先来了解一下报告软件缺陷的基本原则。

（1）尽快报告软件缺陷。软件缺陷发现得越早，则在软件开发过程中留下的修正时间越多，被修正的可能性越大，花费的代价就越少。图 7-12 显示了时间和缺陷修复之间的关系。

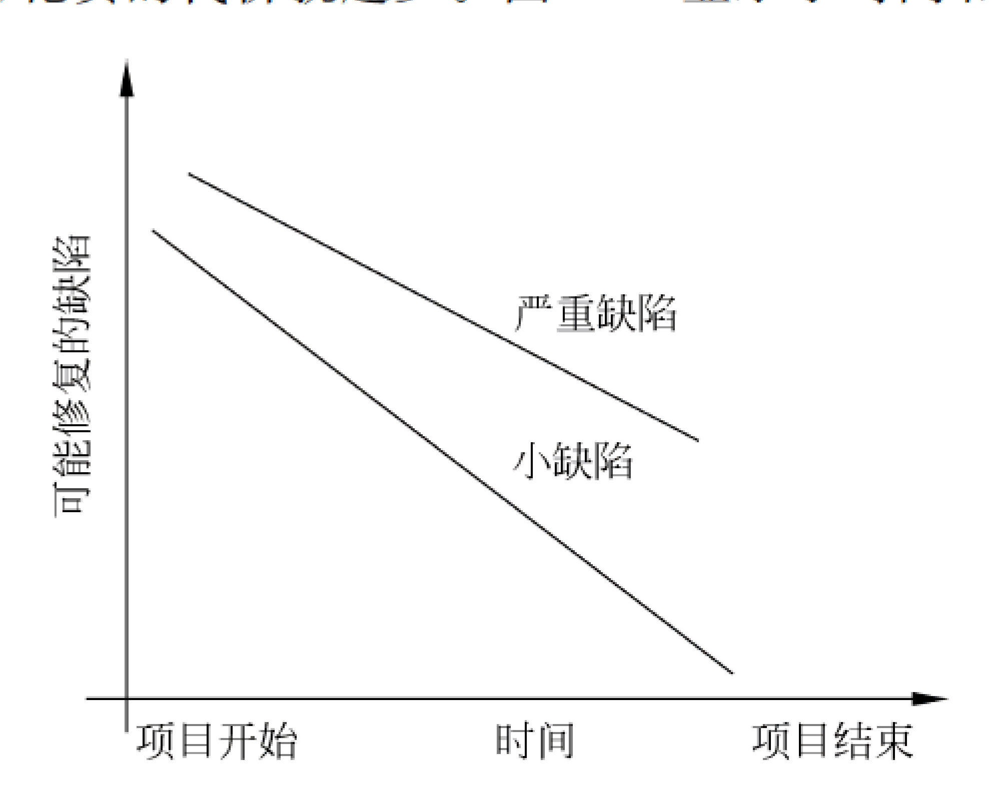


图 7-12　软件缺陷发现得越晚，越不可能被修复

（2）有效地描述软件缺陷。测试人员在提交缺陷报告时，应站在开发人员的角度上思考问题，要确保开发人员拿到缺陷报告后马上就能明白问题，而不会产生理解上的歧义。假如你是开发人员，看到如下所述的报告：“无论何时在登录对话框中输入一串随机字符，软件就会进行一些奇怪的动作”，你肯定会想：随机字符是什么，有多长，会产生什么奇怪的动作，应如何着手去修复这个缺陷呢？

有效的软件缺陷描述如下所示：

① 简单，短小。只解释事实和演示、描述软件缺陷必需的细节，揭示错误的实质。

② 单一。每个缺陷报告只针对一个软件缺陷。

③ 使用 IT 业界惯用的表达术语和表达方式。

④ 明确指明错误类型。

（3）在报告软件缺陷时不做任何评价。由于测试人员的责任是找出软件中存在的缺陷，这样测试人员和开发人员之间很容易形成对立关系，因此在提交缺陷报告时要保持中立的态度，不要带有倾向性、个人观点和煽动性，例如：“这个问题在上一个版本中已经修复了，怎么这次又出现了呢？”“这个错误太低级了”等，这些语句都不应出现在缺陷报告中。

（4）确保缺陷可以重现。如果开发人员不能重现你提交的缺陷，将会影响开发人员的开发效率，也会影响测试人员自身的声誉，因此在提交缺陷之前一定要确保缺陷能够重现；对于严重程度较高的缺陷，一般要能保证按照预定步骤可以使其重复测试两次以上，对于随机产生的缺陷，要在其他机器上测试一下，看是否是自己机器的问题。

测试人员应该牢记上面这些关于报告软件缺陷的原则。这些原则几乎可以运用到任何交流活动中，尽管有时难以做到，但如果希望有效地报告软件缺陷，并使其得以修复，测试人员就必须遵循这些基本原则。

3. 缺陷报告的读者

缺陷报告的直接读者是软件开发人员和质量管理人员，来自市场和技术支持等部门的相关人员也可能需要了解缺陷情况。缺陷报告的读者最希望获得的信息包括①软件缺陷报告中

的缺陷；②报告的软件缺陷各自独立，缺陷信息具体、准确；③缺陷的本质特征和复现步骤；④缺陷类型分布以及对市场和用户的影响程度。

7.3.2 缺陷报告撰写标准

软件缺陷报告又称软件问题报告，是软件工程技术规范中的一项重要内容，是软件测试过程中非常重要的文档。它记录了缺陷或问题(Bug)发生的环境，如各种资源的配置情况、Bug 的再现步骤以及 Bug 性质的说明。更重要的是它还记录着 Bug 的处理过程和状态。Bug 的处理进程从一定程度反映了测试的进程和被测软件的质量状况以及改善过程。

软件问题报告的编写或撰写是有要求和规范的，有相应的文档编写标准。如国标、国军标以及行业标准。如图 7-13 给出了我国某行业的软件问题报告编制模板。

附录H 软件问题报告

H.1 编写目录
1 范围
1.1 标识
1.2 系统概述
1.3 文档概述
2 引用文件
3 术语和缩略语
4 软件问题报告
4.1 被评测软件的问题
4.1.1 测试发现的软件问题
4.1.1.1 小结
4.1.1.2 测试问题报告单
4.1.2 检查发现的问题
4.1.2.1 小结
4.1.2.2 检查问题报告单
4.2 其他问题

H.2 编写指南
1 范围
1.1 标识
写明本文档的：
a. 已批准的标识号；
b. 标题；
c. 本文档适用的被评测软件名称。
1.2 系统概述
概述本文档适用的系统和系统的用途。
1.3 文档概述
简要说明本文档的目的和用途。
2 引用文件
按文档号、标题、编写单位（或作者）和出版日期等，列出本文档引用的所有文件。
3 术语和缩略语
本章给出所有在本文档中出现的专用术语和缩略语的确切定义。
4 软件问题报告
4.1 被评测软件的问题
4.1.1 测试发现的软件问题
4.1.1.1 小结
对测试中发现的软件问题进行汇总。
4.1.1.2 测试问题报告单
按照《软件测试细则》的要求，填写“软件问题报告单”。
4.1.2 检查发现的问题
4.1.2.1 小结
对检查中发现的软件问题进行汇总。
4.1.2.2 检查问题报告单
按照《软件测试细则》的要求，填写“软件问题报告单”。
4.2 其他问题
描述评测发现的其他问题。按照《软件测试细则》的要求，填写“软件问题报告单”。

图 7-13 我国某行业的软件问题报告编制模板

按照图 7-13 编制软件问题报告，需要事先填写软件问题报告单。该报告单的填写也有相应的要求、规范或标准。图 7-14 给出了某公司软件问题报告单的填写模板。

软件问题报告单

编号：__________

系统名称及版本					
模块名称及版本					
模块版本信息	（文件创建日期、文件大小、相关其他文件信息等）				
报告人			联系电话		
发生日期			提交日期		
问题类型：	程序□	数据库□	文档□	其他□	
要求完成日期			实际完成日期		
问题描述/影响（问题发现过程与曾经处理过程）：					
问题背景描述： 例如，系统使用方，工程启动时间，工程预期结束时间，工程当前状态（测试、试运行，还是已经投产），系统的版本号，软件的来源方式等。					
问题严重性：□很严重：严重影响系统验收、投产或正常运行。 □严重：影响系统验收、投产或正常运行。 □一般：影响操作。					
问题影响到下一步工作描述：					
受理人		受理时间		承诺解决时间	
受理人意见：					
修改人			完成时间		
问题解决过程以及解决办法描述：					
问题解决结果并分析原因：					
报告人确认并签名：					

图 7-14　软件问题报告单的填写模板

7.4　缺陷管理工具

缺陷管理是软件开发和软件质量管理的重要组成部分，是软件开发管理过程中与配置管理并驾齐驱的最基本管理需求。目前，随着人们对缺陷管理工具的需求逐渐增多而且更加明确，国内外越来越多的公司进行相关管理工具的开发、包括缺陷管理工具的开发、提供高质量的商用工具。同时，人们渴望得到物美价廉的可用版本（当然大多数都有免费的试用板）。缺陷管理及缺陷管理工具的重要性和被重视程度越来越高。

缺陷管理工具用于集中管理软件测试过程中发现的错误，是添加、修改、排序、查寻、存储软件测试错误的数据库程序。

大型本地化软件测试项目一般测试周期较长，测试范围广，存在较多软件缺陷。如果对测

试质量要求较高，并有支持多语言或本地化的要求，特别需要缺陷管理工具。

缺陷管理工具的使用使得查找和跟踪方便了，对于大型本地化软件的测试，报告的错误数目可能成千上万，如果没有缺陷管理工具，要查找某个错误，将是一件痛苦的事。

另外，缺陷管理工具的使用也使得跟踪和监控错误的处理过程和方法更加容易，既可以方便地检查处理方法是否正确，也可以确定处理者的姓名和处理时间，作为统计和考核工作质量的参考。

而且，缺陷管理工具的使用为集中管理提供了支持条件，为大大地提高管理效率提供了可能。如本地化服务商和软件供应商共享同一个缺陷数据库，各自负责处理需要处理的软件错误。如果需要对方提供更多的缺陷信息，则可以通过改变缺陷当前的属性(状态、处理者、处理建议等)，使对方尽快处理。

最后，缺陷管理工具的使用使得整个缺陷管理安全性高，通过权限设置，不同权限的用户能执行不同的操作，保证只有适当的人员才能执行正确的处理；同时，能够保证缺陷处理顺序的正确性，根据当前错误的状态，决定当前错误的处理方法。最重要的是缺陷管理工具具有方便存储的特点，便于项目结束后的缺陷管理活动历史过程存档，可以随时或在项目结束后存储，以备将来参考。

下面是几款国际、国内比较知名的缺陷管理工具的介绍。

7.4.1 TrackRecord(商用)

作为 Compuware 项目管理软件集成的一个重要组成部分，TrackRecord 目前已经拥有众多的企业级用户，它基于传统的缺陷管理思想，整个缺陷处理流程完备，界面设计精细，并且对缺陷管理数据进行了初步的加工处理，提供了一定的图形表示。显著特点如下：

(1) 定义了信息条目类型。在 TrackRecord 的数据库中，定义了不同的缺陷、任务、组成员等内容，它们可通过图形界面进行输入。

(2) 定义规则。规则引擎(Rules Engine)允许管理者对不同信息类型创建不同的规则，规定不同字段的取值范围等。

(3) 工作流程。一个缺陷、任务或者其他条目，从它被输入到最后清除期间经历的一系列状态。

(4) 查询。对历史信息进行查询，显示其查询结果。

(5) 概要统计或图形表示。动态地对数据库中的数据进行统计报告，可按照不同的条件进行统计，同时提供了几种不同的图形显示：①文本方式显示不同缺陷状态、列表；②立体彩色条形图显示不同优先级或不同开发者不同优先级的缺陷状态；③彩色饼图显示所有人员发现缺陷占总缺陷数的百分比。

(6) 网络服务器。网络服务器允许用户通过网络浏览器访问数据库。

(7) 自动电子邮件通知。提供报告的缺陷邮件通知功能，并为非注册用户提供远程视图(在保证项目信息安全的情况下，让某些非项目组人员可以了解项目的相关信息)。

7.4.2 ClearQuest(商用)

IBM Rational 一向以功能强大、产品类型全面而著称。IBM Rational ClearQuest 是基于团队的缺陷和变更跟踪解决方案，是一套高度灵活的缺陷和变更跟踪系统，适用于在任何平台上、任何类型的项目中，捕获各种类型的变更。

它的强大之处和显著特点表现在以下几个方面：

（1）支持 MS Access 和 SQL Server 等数据库。

（2）拥有可完全定制的界面和工作流程机制，能适用于任何开发过程。

（3）可以更好地支持最常见的变更请求（包括缺陷和功能改进请求），并且便于对系统做进一步的定制，以便管理其他类型的变更。

（4）提供了一个可靠的集中式系统，该系统与配置管理、自动测试、需求管理和过程指导等工具集成，使项目中每个人都可以对所有变更发表意见，并了解其变化情况。

（5）与 IBM Rational 的软件管理工具 ClearCase 完全集成，让用户充分掌握变更需求情况。

（6）能适应所需的任何过程、业务规则和命名约定。可以使用 ClearQuest 预先定义的过程、表单和相关规则，或者 ClearQuest Designer 来定制——几乎系统的所有方面都可以定制，包括缺陷和变更请求的状态转移生命周期、数据库字段、用户界面布局、报表、图表和查询等。

（7）强大的报告和图表功能，使用户能直观、简便地使用图形工具定制所需的报告、查询和图表，帮助用户深入分析开发现状。

（8）自动电子邮件通知、无须授权的 Web 登录以及对 Windows、UNIX 和 Web 的内在支持，ClearQuest 可以确保团队中的所有成员，都被纳入缺陷和变更请求的流程中。

7.4.3 Bugzilla（开源）

Bugzilla 是一个“缺陷跟踪系统”或者“bug 跟踪系统”，帮助个人或者小组开发者有效地跟踪已经发现的错误。与多数商业缺陷管理软件收取昂贵的授权费用相比，Bugzilla 作为一个开源免费软件，拥有许多商业软件所不具备的特点，因而，现在已经成为全球许多组织喜欢的缺陷管理软件。它的主要特点如下。

（1）普通报表生成：自带基于当前数据库的报表生成功能。

（2）基于表格的视图：一些图形视图（条形图、线性图、饼图）。

（3）请求系统：可以根据复查人员的要求对 bug 进行注释，以帮助他们理解并决定是否接受该 bug。

（4）支持企业组成员设定：管理员可以根据需要定义由个人或者其他组构成的访问组。

（5）支持用户名通配符匹配功能：当用户输入一个不完整的用户名时，系统会显示匹配的用户列表。

（6）内部用户功能：可以定义一组特殊用户，他们所发表的评论和附件只能被组内成员访问。

（7）时间追踪功能：系统自动记录每项操作的时间，并显示规定结束时间的剩余时间。

（8）多种验证方法：模型化的验证模块，使用户方便地添加所需系统验证。Bugzilla 已经内建了支持 MySQL 和 LDAP 授权验证的方法。

（9）本地化配置：管理员可以根据用户所在地域而自动使用当地用户的字体进行页面显示。

（10）补丁阅读器：增强了与 Bonsai、LXR 和 CVS 整合过程中提交的补丁的阅读功能，为设计人员提供丰富的上下文。

（11）评论回复连接：对 bug 的评论提供直接的页面连接，帮助复查人员评审 bug。

（12）支持数据库全文检索，包括对评论、概括等。

（13）E-mail 地址加密，保护使用者的电子邮件地址不被非法获取。

（14）视图生成功能：高级的视图特性允许用户在可配置数据集的基础上灵活地显示数据。

（15）统一性检测：扫描数据库的一致性。报告错误并允许客户打开与错误相关的 bug 列表。统一性检测同时检测用户的发送邮件列表，提示未发送邮件队列等的状态。

7.4.4 BMS(国内商业软件)

BMS是上海微创软件有限公司(由上海联和投资有限公司与全球软件行业领头羊微软公司合资成立的新兴软件企业)推出的软件开发管理解决方案的核心产品,将微软丰富的项目开发经验与众多用户的实际需求结合起来,帮助中小软件企业规范和完善管理流程、强化产品质量,并从根本上推动企业管理思想和方法的进步。BMS的主要特点如下:

(1) 在微软.NET技术的基础上,BMS可以全方位地跟踪、管理、统计和分析企业内部项目质量管理过程中的缺陷,最大限度减少缺陷的出现率,进而实现软件质量的量化。

(2) BMS可记录企业软件开发过程中发现的缺陷,提供不同条件的缺陷查询与针对性管理。

(3) 能够以多种形式的统计报表帮助相关人员直观地掌握缺陷的全局情况,实现对整个软件开发过程的多层次、多角度管理和对软件开发的状况与进度进行宏观调控。

(4) 具有决策支持、实时通知等实用性功能,也对软件企业工作效率的提高和流程的改善助益良多。

(5) 良好的跨平台能力,无论客户从事的是通用软件产品开发、项目定制,还是硬件相关的集成开发,都可作为BMS的用武之地。这一点在国内千差万别的复杂软件开发环境中,有着格外重要的意义。

7.4.5 其他

1. Buggit(开源)

Buggit是一个十分小巧的C/S结构的Access应用软件,仅限于Intranet,十分钟就可以配置完成,使用十分简单,查询简便,能满足基本的缺陷跟踪功能,还有十个用户定制域,有十二种报表输出。

2. Mantis(开源)

Mantis(开源)是一款基于Web的软件缺陷管理工具,配置和使用都很简单,适合中小型软件开发团队。

3. HP Quality Center(简称QC,商用)中的缺陷管理模块

HP QC是一个基于Web的测试管理工具,可以组织和管理应用程序测试流程的所有阶段,包括指定测试需求、计划测试、执行测试和跟踪缺陷。QC中一个重要的功能就是缺陷的管理,主要是规范缺陷在其生命周期各个阶段中正常的操作,如缺陷的状态修改、权限控制(用户在操作不同状态下的缺陷时受到权限的影响)等。

习题

1. 什么是软件缺陷?一般是如何描述和分类软件缺陷的?
2. 简述缺陷的来源与影响,分析缺陷是在软件开发生命周期的哪个阶段产生的。
3. 什么是软件缺陷管理,缺陷管理报告单包括哪些内容?具有什么特点?
4. 根据自己的理解,画出软件缺陷管理流程图,并解释软件缺陷管理流程图的关键要素。
5. 如何度量软件缺陷?如何分析软件缺陷?如何对软件缺陷进行统计?
6. 简述软件缺陷描述中的缺陷基本信息和软件缺陷分类中的缺陷属性。
7. 软件缺陷报告所包含的主要内容有哪些?其撰写标准主要有哪些?
8. 简述3款缺陷管理工具的功能和特点,并进行比较。

第8章 软件测试过程及测试过程管理

通过对前面章节的学习，了解到软件测试不能等待编码完成后才开始对程序进行测试，而是在项目需求分析阶段就要参与进去，审查需求分析文档、产品规格说明书；然后在设计阶段，要审查系统设计文档、程序设计流程图、数据流图等；在代码测试阶段，需要审查代码，看是否遵守代码的变量定义规则、是否有足够的注释行等。因此，从软件开发生命周期角度看，软件测试也存在着生命周期概念，即软件测试生命周期。软件测试生命周期一般包括7个阶段：计划、分析、设计、构建、测试用例编写、最后测试实施以及测试总结。测试的生命周期是软件生命周期的一部分，应该与开发周期同时开始，而且软件测试周期的各个阶段应该和开发周期的各个阶段一一对应。测试的过程与应用程序的开发过程一样复杂和艰巨。如果未能尽早开始，会导致在开发时间表上附加一个长时间、高成本的测试和错误修正时间表。

软件测试过程与软件开发过程一样，可用一种抽象模型表示，用于定义软件测试的流程和方法，指导测试团队按照承担软件测试项目所要求的进度、成本和质量，开展测试任务必须覆盖整个软件测试生命周期的一组有序的软件测试活动。众所周知，软件开发过程的质量决定了软件的质量，同样的，软件测试过程的质量将直接影响测试结果的准确性和有效性。软件测试过程和软件开发过程一样，都遵循软件工程原理，遵循管理学原理。因此，一般意义上的软件测试过程包括软件测试需求分析、测试计划制定、测试用例设计、测试工具开发、测试环境构建、测试实施及测试评估等阶段，每个阶段都有一系列的任务。

此时，软件测试不再只是对程序代码的测试工作，而是包含了所有在软件开发过程阶段产出物的测试工作，这扩展了测试工作的范围，有利于发现和控制软件开发中的缺陷。

另外，软件开发与软件测试应该是交互进行的。例如，单元编码需要单元测试，模块组装阶段需要集成测试。如果等到软件编码结束后才进行测试，那么，测试的时间将会很短，测试的覆盖面将不全面，测试的效果也将打折扣。更严重的是如果此时发现了软件需求阶段或概要设计阶段的错误，要修复该类错误，将会耗费大量的时间和人力。

而且，软件测试过程清晰地定义了过程的输入输出流，为当前整个测试过程实施的度量和今后测试过程的改进奠定了基础。

最后，要强调将测试的结果纳入到软件开发的缺陷管理机制中，扩展测试的作用。

8.1 软件测试过程

软件测试过程要求基于项目的整体需求，对整个测试生命周期中的所有过程、活动及变更进行定义、控制和管理。在当前基于软件生命周期的开发过程中，人们常用到的主流的软件生命周期模型或软件开发过程模型有瀑布模型、原型模型、螺旋模型、增量模型、渐进模型、快速软件开发(RAD)以及Rational统一过程(RUP)等。这些模型对软件开发过程具有很好的指

导作用，但是在这些过程方法中，软件测试的地位和价值并没有体现出来，也没有给软件测试以足够的重视，利用这些模型无法更好地指导测试实践。软件测试是与软件开发紧密相关的一系列有计划、系统性的活动，显然软件测试也需要测试模型去指导实践。

8.1.1 软件测试过程模型

软件测试模型的研究随着软件工程的发展而越来越深入。在 20 世纪 80 年代后期，Paul Rook 提出了著名的软件测试的 V 模型，旨在改进软件开发的效率和效果。

1. V 模型

在传统的开发模型中，如瀑布模型，通常把测试过程作为在需求分析、概要设计、详细设计和编码全部完成之后的一个阶段，尽管有时测试工作会占用整个项目周期一半的时间，但是有人仍认为测试只是一个收尾工作，而不是主要的工程。V 模型是软件开发瀑布模型的变种，它反映了测试活动与分析和设计的关系，从左到右，描述了基本的开发过程和测试行为，明确地表明了测试过程中存在不同类型、不同级别的测试，清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系，如图 8-1 所示。

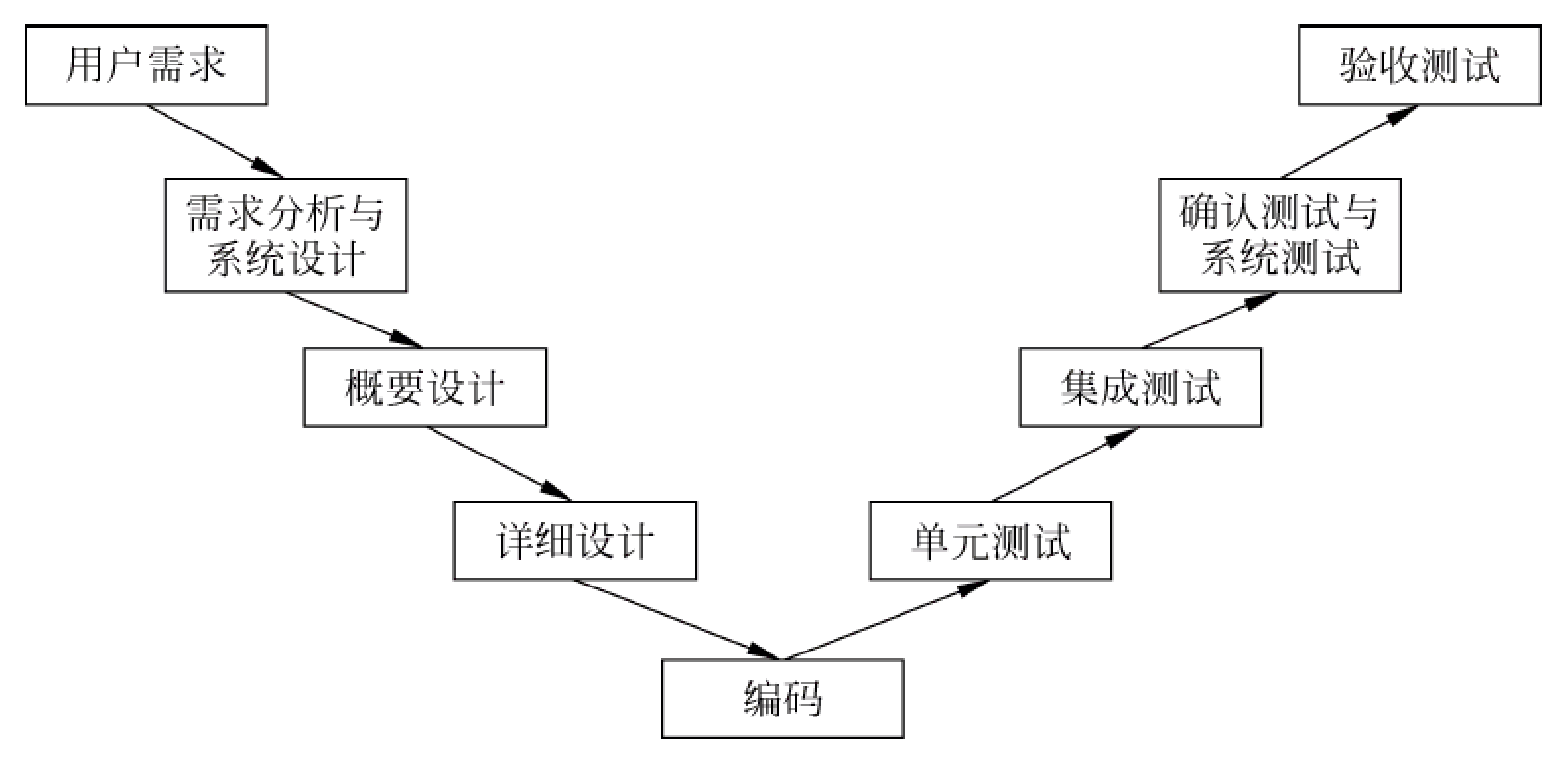


图 8-1 V 模型图

图 8-1 所示的 V 模型图中箭头代表了时间方向，左边下降的是开发过程各阶段，与此相对应的是右边上升的部分，即测试过程的各个阶段。

V 模型的软件测试策略既包括低层测试又包括了高层测试，低层测试是为了源代码的正确性，高层测试是为了使整个系统满足用户的需求。

V 模型指出，单元和集成测试是验证程序设计，开发人员和测试组应检测程序的执行是否满足软件设计的要求；系统测试应当验证系统设计，检测系统功能、性能的质量特性是否达到系统设计的指标；由测试人员和用户进行软件的确认测试和验收测试，追溯软件需求说明书进行测试，以确定软件的实现是否满足用户需求或合同的要求。

V 模型存在一定的局限性，它仅仅把测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段。容易使人理解为测试是软件开发的最后的一个阶段，主要是针对程序进行测试寻找错误，而需求分析阶段隐藏的问题一直到后期的验收测试才被发现。

2. W 模型

V 模型的局限性在于没有明确地说明早期的测试，不能体现“尽早地和不断地进行软件测试”的原则。在 V 模型中增加软件各开发阶段应同步进行的测试，被演化为一种 W 模型，因为实际上开发是 V，测试也是与此相并行的 V，如图 8-2 所示。

相对于 V 模型，W 模型更科学。W 模型可以说是 V 模型自然而然的发展。它强调：测

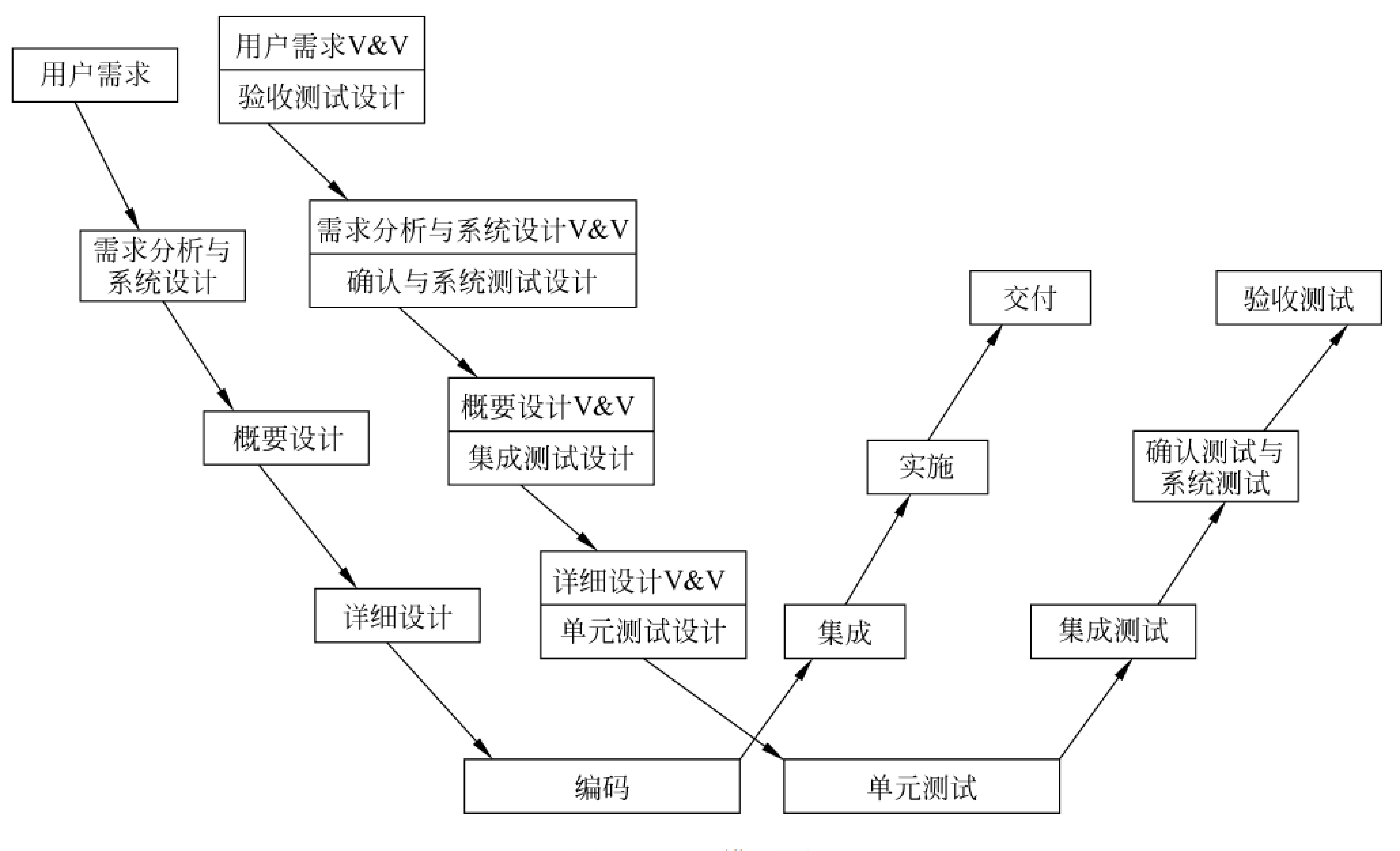


图 8-2　W 模型图

试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、功能和设计同样要测试。这样，只要相应的开发活动完成，我们就可以开始执行测试。可以说，测试与开发是同步进行的，从而有利于尽早地发现问题。以需求为例，需求分析完成后，测试人员就应该参与到对需求的验证和确认活动中，以尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，以减少总体测试时间，加快项目进度。

如果测试文档能尽早提交，那么就有了更多的检查和检阅的时间，这些文档还可用于评估开发文档。另外，还有一个很大的益处是，测试者可以在项目中尽可能早地面对规格说明书中的挑战。这意味着测试不仅仅是评定软件的质量，测试还可以尽可能早地找出缺陷所在，从而帮助改进项目内部的质量。参与前期工作的测试者可以预先估计问题和难度，这将可以显著地减少总体测试时间，加快项目进度。

根据 W 模型的要求，一旦有文档提供，就要及时确定测试条件，以及编写测试用例，这些工作对测试的各级别都有意义。当需求被提交后，就需要确定高级别的测试用例来测试这些需求。当概要设计编写完成后，就需要确定测试条件来查找该阶段的设计缺陷。

W 模型也是有局限性的。W 模型和 V 模型都把软件的开发视为需求、设计、编码等一系列串行的活动。同样，软件开发和测试保持一种线性的前后关系，需要有严格的指令表示上一阶段完全结束，才可以正式开始下一个阶段。这样就无法支持迭代、自发性以及变更调整。对于当前很多文档需要事后补充，或者根本没有文档的做法（这已成为一种开发的文化），开发人员和测试人员都面临同样的困惑。

3. H 模型

V 模型和 W 模型均存在一些不妥之处。首先，如前所述，它们都把软件的开发视为需求、设计、编码等一系列串行的活动，而事实上，虽然这些活动之间存在相互牵制的关系，但在大部分时间内，它们是可以交叉进行的。虽然软件开发期望有清晰的需求、设计和编码阶段，但实践告诉我们，严格的阶段划分只是一种理想状况。试问，有几个软件项目是在有了明确的需求之后才开始设计的呢？所以，相应的测试之间也不存在严格的次序关系。同时，各层次之间的

测试也存在反复触发、迭代和增量关系。其次，V 模型和 W 模型都没有很好地体现测试流程的完整性。

为了解决以上问题，提出了 H 模型。它将测试活动完全独立出来，形成一个完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来。H 模型如图 8-3 所示。

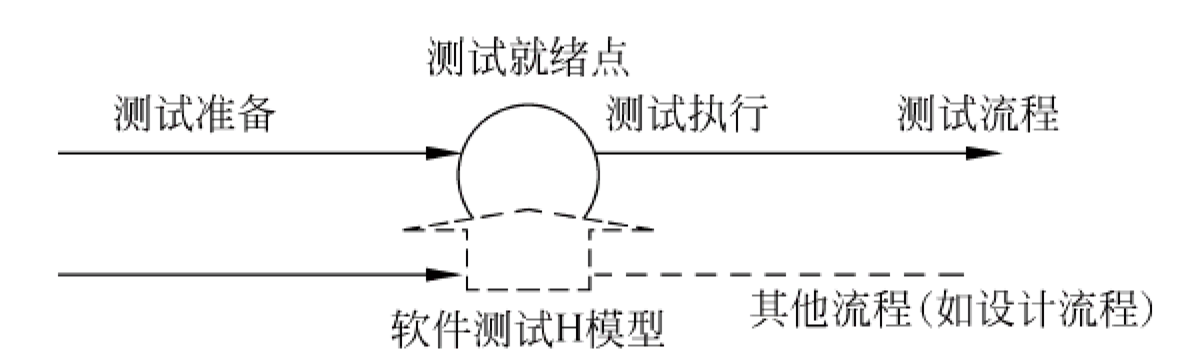


图 8-3　H 模型图

H 模型图仅仅演示了在整个生产周期中某个层次上的一次测试“微循环”。图中的其他流程可以是任意开发流程，例如，设计流程和编码流程。也可以是其他非开发流程，例如，SQA 流程，甚至是测试流程自身。也就是说，只要测试条件成熟了，测试准备活动完成了，测试执行活动就可以(或者说需要)进行了。

概括地说，H 模型揭示了：①软件测试不仅仅指测试的执行，还包括很多其他的活动；②软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行；③软件测试要尽早准备，尽早执行；④软件测试是根据被测物的不同而分层次进行的。不同层次的测试活动可以是按照某个次序先后进行的，但也可能是反复的。

在 H 模型中，软件测试模型是一个独立的流程，贯穿于整个产品周期，与其他流程并发地进行。当某个测试时间点就绪时，软件测试即从测试准备阶段进入测试执行阶段。

4. 其他测试模型

除了上述三种常见模型外，还有其他几种模型，如 X 模型、前置测试模型等。X 模型提出针对单独的程序片段进行相互分离的编码和测试，然后通过频繁的交接，通过集成最终合成为可执行的程序。这些可执行程序还需要进行测试。已通过集成测试的部件可以进行打包并提交给用户，也可以作为更大规模和范围内集成的一部分。

前置测试模型体现了开发与测试的结合，要求对每个交付的内容进行测试。这些模型都针对其他模型的缺点进行了一些修正，但本身仍然存在一些不足的地方。所以在软件测试过程中正确选取模型是一个很关键的问题。

5. 测试模型的使用

前面介绍的几种典型的测试模型，应该说它们对指导测试工作很有意义，但任何模型都不是完美的。我们应该尽可能地去应用模型中对项目有实用价值的方面，但不强行地为使用模型而使用模型，否则也没有实际意义。

在这些模型中，V 模型强调了在整个软件项目开发中需要经历的若干个测试级别，而且每一个级别都与一个开发级别相对应，但它忽略了测试的对象不应该仅仅只有程序，或者说它没有明确地指出应该对软件的需求、设计进行测试，而这一点在 W 模型中得到了补充。W 模型强调了测试计划等工作的先行和对系统需求及系统设计的测试，但 W 模型和 V 模型一样也没有专门对软件测试流程予以说明。因为事实上，随着软件质量要求越来越为大家所重视，软件测试也逐步发展成为一个独立于软件开发的一系列活动，就每一个软件测试的细节而言，它都有一个独立的操作流程。例如，现在的第三方测试，就包含了从测试计划和测试用例编写，到测试实施以及测试报告编写的全过程，这个过程在 H 模型中得到了相应的体现，表现为测试是独立的。也就是说，只要测试前提具备了，就可以开始进行测试了。

因此，在实际的工作中，我们要灵活地运用各种模型的优点，在W模型的框架下，运用H模型的思想进行独立的测试，并同时将测试与开发紧密结合，寻找恰当的就绪点开始测试并反复迭代测试，最终保证按期完成预定目标。

8.1.2 软件测试过程中的活动及内容

软件测试贯穿整个软件开发周期，从需求提出到最终软件提交，软件测试过程中的关键活动包括提取测试需求、确定测试策略、制定测试计划、开展测试设计、执行测试用例、分析测试结果等。测试过程的主要内容是：①基于项目目标，制定测试计划，确定测试策略，选定测试方法，排定优先级，建立里程碑，组织测试资源（测试团队、软硬件环境等）等；②以测试计划为基础，明确测试需求、测试对象和测试目标及功能与性能指标；③依据测试计划和测试设计，测试人员可以开展测试的相关活动（如开发和设计测试用例，选定测试工具，设计自动化测试框架，编写测试脚本，执行测试用例及测试脚本，分析测试结果，发现和报告缺陷）。下面是具体的测试活动及内容。

1. 需求与规范管理（需求阶段）

需求分析阶段的工作就是对用户提出的需求进行分析并将每项分析结果作为测试设计阶段的输入。在该阶段，测试所要实现的目标就是确定测试需求。

（1）由需求人员确定规范和需求，将规范和需求转发给开发经理、项目经理、相关开发和测试人员。

（2）相关需求人员、项目经理对需求进行讨论，整理出重点，并将重点内容发给开发经理及项目经理、相关开发和测试人员。

（3）对需求进行讨论，确定需求指标以及最终实现的需求和功能点。

（4）项目经理根据需求和开会讨论结果编写“需求说明”，估算开发工作量。测试负责人或需求负责人对文档进行检查并修复完善。

（5）测试负责人根据项目的“需求说明”来确定测试的需求，估算测试工作量。

2. 项目计划与测试计划（产品设计阶段）

根据开发估算的工作量进行项目计划，由项目经理给出计划表。然后，项目经理组织项目计划讨论会。讨论过程中，各开发负责人（包括测试负责人）对自己负责的模块或工作所需要的工作量进行评估，根据工作量和工程需求初步确定总体开发计划、测试计划和发布时间，项目经理根据工作量和需求制定项目计划。

（1）由开发经理组织讨论会，各模块负责人评估工作量，根据工作量和需求初步确定开发计划、测试计划和发布时间。

（2）项目经理根据估算工作量和需求编写项目计划。

（3）测试负责人根据估算工作量和需求编写测试计划。

（4）项目计划与测试计划完成后发送给开发经理、项目经理、相关开发人员和测试人员，认真阅读后将意见以邮件形式反馈给项目经理与测试负责人，进行二次修改，修改后召开小型会议进行讨论，最后定稿。

（5）测试负责人确认所有相关文档已得到了评审。

3. 开发设计与评审（产品设计阶段）

该项工作以软件开发人员为主，测试人员可以参与其中了解被测软件的设计情况。

4. 测试方案与评审（产品设计阶段）

（1）项目设计阶段，测试负责人根据规范、功能列表和概要文档编写测试方案。

(2) 测试方案完成后,邮件发送给开发经理、项目经理、相关开发人员和测试人员。

(3) 对测试方案进行评审,最后将意见反馈给测试负责人修改,最终确定测试方案。

5. 测试设计与评审(开发阶段)

(1) 进行详细的测试用例设计,包括功能测试、性能测试、压力测试等,以便发现更多的隐藏问题。

(2) 测试用例设计完成后,各负责人对测试用例进行评审。

6. 编码实现与单元测试(开发阶段、测试阶段)

开发人员编写代码,同时测试人员编写测试用例。

(1) 产品设计完成后,开发工程师进行编码。

(2) 编码完成后,测试工程师编写单元测试用例进行单元测试。

(3) 项目经理根据实际情况对开发的编码组织代码复审,记录相关问题。

(4) 单元测试完成后,进行单元的集成及产品联调测试,并修改发现的问题。

7. 测试实施(测试阶段)

(1) 测试环境中根据测试用例执行测试,在测试过程中发现问题并填写测试记录。

(2) 提交 Bug。

(3) 开发人员修改 Bug。

(4) 修改 Bug 后,测试人员进行回归测试,直至关闭 bug。

8. 产品发布

当测试产品达到测试计划所制定的产品质量目标和测试质量目标,进行最后一轮的确认测试,编写总体测试报告和性能测试报告,确认测试完成后进行产品发布。

8.1.3 软件测试过程度量

软件测试是保证软件质量的重要方法。随着软件开发规模的增大、复杂程度的增加,以寻找软件中的错误为目的的测试工作就更加复杂和困难。因此,为了尽可能多地找出程序中的错误,生产出高质量的软件产品,加强对测试工作的组织和管理就显得尤为重要,而其中很重要的一个方面是进行软件测试过程的度量。软件测试过程度量对于改进软件测试过程,提高软件测试效率具有重要意义。

1. 软件测试过程度量意义

随着软件生产规模的日益增大,保证软件产品质量的软件测试工作越来越得到人们的重视,为更好地保证软件产品质量、更有效地执行软件测试工作,迫切需要对软件测试过程进行有效管理并逐步改善。目前,软件生产过程的管理模式也经历了完全依据管理者的经验到以数据为依据的量化管理这样一个逐步转化的过程。要对软件测试过程进行有效的管理就需要有反映过程本质特性的过程数据,通过这些数据,测试过程的执行状况才能得到监控,测试过程改进的决策才能有的放矢。由于软件测试过程的不可见性,软件测试过程度量成为对软件测试过程进行量化管理不可或缺的一个环节。

软件测试过程度量是这样一种技术,它提取软件测试过程中可计量的属性,在测试过程进行中以一定频度不断地采集这些属性的值,并采用一些恰当的分析方法对得到的这些数据进行分析,建立可度量的指标,从而量化地评定测试过程的能力和性能,提高测试过程的可视性,帮助软件组织管理和改进软件测试过程。

2. 软件测试过程度量指标

软件测试阶段的过程度量内容或项目比较多,包括软件测试进度、测试覆盖度、测试缺陷

出现/到达曲线、测试缺陷累积曲线、测试效率等。在进行测试过程度量时，要基于软件规模度量(如功能点、对象点等)、复杂性度量、项目度量等方法，从测试广度、测试深度以及缺陷数三个不同的测度来完整度量测试过程。测试广度的测度提供了多少需求(在所有需求的数目中)在某一时刻已经被测试，来度量测试计划的执行、测试进度等状态；测试深度是对被测试覆盖的独立基本路径占在程序中的基本路径的总数的百分比的测度，基本路径数目的度量可以用McCabe圈复杂度来计算；过程中收集的缺陷数，以及发现的、修正的和关闭的缺陷数量在过程中的差异、发展趋势等的度量，为开发过程质量、开发资源额外投入、软件发布预测等提供重要依据。

在CMMI4体系的测试过程中定义了4个度量指标：测试覆盖率、测试执行率、测试执行通过率、测试缺陷解决率。下面详细介绍这4个指标的含义及数据收集方法。

1）测试覆盖率

测试覆盖率是指测试用例对需求的覆盖情况。计算公式：

$$\text{测试覆盖率}=\frac{\text{已设计测试用例的需求数}}{\text{需求总数}}$$

测试覆盖率从范围上说包括广度覆盖和深度覆盖；从内容上说包括用户场景覆盖、功能覆盖、功能组合覆盖、系统场景覆盖。

所谓广度，其含义是需求规格说明书中的每个需求项是否都在测试用例中得到设计。而深度，通俗地说，是不使我们的测试设计流于表面，是否能够透过客户需求文档，挖掘出可能存在问题的地方。例如，重复单击某个按钮10次，或者依次执行新增、删除、新增同一数据的记录、再次删除该记录操作。

在设计测试用例时，我们很少单独设计广度或深度方面的测试用例，而一般是结合在一起设计。为了从广度和深度上覆盖测试用例，我们需要考虑设计各种测试用例，如用户场景(识别最常用的20%的操作)、功能点、功能组合、系统场景、性能、语句、分支等。在执行时，需要根据测试时间的充裕程度按照一定的顺序执行。通常是先执行用户场景的测试用例，然后再执行具体功能点、功能组合的测试。

测试覆盖率数据的收集，我们可以通过需求跟踪矩阵来实现。在需求跟踪矩阵，测试人员填写的“系统测试用例”列的数据。测试人员通过计算需求跟踪矩阵列出的需求数量，和已设计测试用例的需求数量，可以快速地计算出测试覆盖率。通过需求跟踪矩阵，测试人员，包括项目组成员都可以很清楚地、快速地知道当前这个项目测试的测试覆盖情况。

2）测试执行率

执行率，顾名思义，就是指实际执行过程中确定已经执行的测试用例比率。计算公式：

$$\text{测试执行率}=\frac{\text{已执行的测试用例数}}{\text{设计的总测试用例数}}$$

通常，我们设计的测试用例一般都是要执行的，即使是按模块来执行测试，那该模块的测试执行率也一般是100%，那么设置这个指标有何意义？

在实际测试过程中，经常有这两种情况发生：①因为系统采用迭代方式开发，每次Build时都有不同的重点，包含不同的内容；②由于测试资源的有限，不可能每次将所有设计的测试内容都全部测试完毕。由于这两种情况的存在，所以在每次执行测试时，我们会按照不同的测试重点和测试内容来安排测试活动，所以就存在了“测试执行率”这个指标。

通常，我们的测试目标是确保100%的测试用例都得到执行，即执行率为100%。但是，如前面所提到的，实际中可能存在非100%的执行率。如果不能达到100%的测试执行率，那么我们需要根据不同的情况制定不同的测试执行率标准——主要考虑风险、重要性、可接受的测

试执行率。在考虑可接受的测试执行率时，就涉及测试用例执行顺序的问题。

在设计测试用例时，需要从广度和深度上尽可能地覆盖需求，所以就需要设计各种测试用例，如正常的测试用例、异常的测试用例、界面的测试用例等。但是在执行时，测试人员需要根据项目进度和测试时间的充裕程度，参考测试执行率标准，将测试用例按照一定的顺序执行。通常是先执行用户场景对应的测试用例，然后再执行具体功能点、功能组合的测试，完成这些测试后，再进行其他测试，如系统场景、性能、语句等测试。

例如，某项目共设计了 280 个测试用例。该项目某一阶段的测试共分 4 个版本，其中有一个版本执行了 134 个测试用例，那么该版本的测试执行率为 47.9%。

3）测试执行通过率

测试执行通过率，指在实际执行的测试用例中，执行结果为“通过”的测试用例比率。计算公式：

$$测试执行通过率=\frac{执行结果为“通过”的测试用例数}{实际执行的测试用例总数}$$

可以针对所有计划执行的测试用例进行衡量，可以细化到具体模块，用于对比各个模块的测试用例执行情况。

为了得到测试执行通过率数据，在测试执行时，需要在测试用例副本中记录下每个测试用例的执行结果，然后在当前版本执行完毕，或者定期（如每周）统计当前测试执行数据。通过原始数据的记录与统计，我们可以快速地得到当前版本或当前阶段的测试执行通过率。

4）测试缺陷解决率

缺陷解决率，指某个阶段已关闭缺陷占缺陷总数的比率。缺陷关闭操作包括以下两种情况：①正常关闭。缺陷已修复，且经过测试人员验证通过。②强制关闭。诸如重复的缺陷、由于外部原因造成的缺陷、暂时不处理的缺陷、无效的缺陷等。这类缺陷经过确认后，可以强制关闭。计算公式：

$$缺陷解决率=\frac{已关闭的缺陷}{缺陷总数}$$

在项目过程中，在开始时缺陷解决率上升很缓慢，随着测试工作的开展，缺陷解决率逐步上升，在版本发布前，缺陷解决率将趋于 100%。一般来说，在每个版本对外发布时，缺陷解决率都应该达到 100%。也就是说，除了已修复的缺陷需要进行验证外，其他需要强制关闭的缺陷必须经过确认，且有对应的应对措施。可以将缺陷解决率作为测试结束和版本发布的一个标准。如果有部分缺陷仍处于打开或已处理状态，那么原则上来说，该版本是不允许发布的。

可以通过缺陷跟踪工具定期（如每周）收集当前系统的缺陷数、已关闭缺陷数，通过这两个数据，即可绘制出整个项目过程或某个阶段的缺陷解决率曲线。

实际上，测试度量指标不仅仅只包括上述 4 个，在实际工作中，还会用到如验证不通过率、缺陷密度等指标。收集这些数据目的是能对测试过程进行量化管理。但是，简单收集度量数据不是目的，通过对数据的分析，达到预防问题、对问题采取纠正措施、减少风险才是目的。

3. 软件测试过程度量原则

对软件测试过程质量度量应该遵循四项原则：①要制定明确的度量目标；②建立软件测试过程质量度量的指标体系，度量指标的定义应该具有一致性、客观性；③度量的方法应该尽可能简单、可计算；④度量数据的收集应该尽可能自动化。

8.1.4 软件测试过程成熟度

从本质上来说，无论是传统的软件测试，还是面向整个开发过程的基于生命周期的软件

测试，其所针对的测试对象都是软件产品、半成品或者过程工作产品，其所报告的测试结果也只是为了识别出现阶段产品的缺陷，并加以纠正以支持下一阶段的开发工作。

从软件开发组织的长远发展来看，仅仅做到这些还是不够的。软件测试作为软件质量保证的一种重要手段，不仅要能够识别软件产品的缺陷并加以改正，还应该在软件测试中结合统计技术方法，给出对软件开发过程的度量，从而支持组织对软件开发过程的评估和改进。由美国国防部和卡耐基-梅隆大学的软件工程研究所联合开发的软件能力成熟度模型 CMM，正是从软件过程改进和评估的角度出发，对软件开发中的测试技术给出了充分的支持和扩充。但仅仅如此是不够的，因为在 CMM 中，软件测试并没有予以充分定义，软件测试成熟度概念没有提及，软件测试过程的改进没有充分说明，在关键过程域 KPA 中没有定义测试问题，与质量相关的测试问题如可测性、充分测试标准、测试计划等方面也没有满意的阐述。仅在第三级的软件产品工程(SPE)KPA 中提及软件测试职能，但对于如何有效提高机构的测试能力和水平没有提供相应指导，这显然是有问题的。

为此，许多研究机构和测试服务机构从不同角度出发提出有关软件测试方面的能力成熟度模型，作为 SEI-CMM 的有效补充，比较有代表性的包括美国国防部提出一个 CMM 软件评估和测试 KPA 建议；Gelper 博士提出一个测试支持模型(TSM)，用于评估测试小组所处的环境及对测试的支持程度；Burgess/Drabick I. T. I. 公司提出的测试能力成熟度模型(Testing Capability Maturity Model)则提供了与 CMM 完全一样的 5 级模型。Burnstein 博士提出了测试成熟度模型(TMM)，依据 CMM 的框架提出测试的 5 个不同级别，关注于测试的成熟度模型。它描述的测试过程，是项目测试部分得到良好计划和控制的基础。TMM 测试成熟度分解为 5 级别，关注于 5 个成熟度级别递增。

1. 初始级

TMM 初始级软件测试过程的特点是测试过程无序，有时甚至是混乱的，几乎没有妥善定义的。初始级中软件的测试与调试常常被混为一谈，软件开发过程中缺乏测试资源、工具以及训练有素的测试人员。初始级的软件测试过程没有定义成熟度目标。

2. 定义级

TMM 的定义级中，测试已具备基本的测试技术和方法，软件的测试与调试已经明确地被区分开。这时，测试被定义为软件生命周期中的一个阶段，它紧随在编码阶段之后。但在定义级中，测试计划往往在编码之后才得以制定，这显然有悖于软件工程的要求。

TMM 的定义级中需实现 3 个成熟度目标：制定测试与调试目标，启动测试计划过程，制度化基本的测试技术和方法。

1）制定测试与调试目标

软件组织必须清晰地区分软件开发的测试过程与调试过程，识别各自的目标、任务和活动。正确区分这两个过程是提高软件组织测试能力的基础。与调试工作不同，测试工作是一种有计划的活动，可以进行管理和控制。这种管理和控制活动需要制定相应的策略和政策，以确定和协调这两个过程。

制定测试与调试目标包含 5 个子成熟度目标：①分别形成测试组织和调试组织，并有经费支持；②规划并记录测试目标；③规划并记录调试目标；④将测试和调试目标形成文档，并分发至项目；⑤将测试目标反映在测试计划中。

2）启动测试计划过程

制定计划是使一个过程可重复、可定义和可管理的基础。测试计划应包括测试目的、风险分析、测试策略以及测试设计规格说明和测试用例。此外，测试计划还应说明如何分配测试资

源，如何划分单元测试、集成测试、系统测试和验收测试的任务。启动测试计划过程包含5个子目标：①建立组织内的测试计划组织并予以经费支持；②建立组织内的测试计划政策框架并予以管理上的支持；③开发测试计划模板并分发至项目的管理者和开发者；④建立一种机制，使用户需求成为测试计划的依据之一；⑤评价、推荐和获得基本的计划工具并从管理上支持工具的使用。

3）制度化基本的测试技术和方法

为改进测试过程能力，组织中需应用基本的测试技术和方法，并说明何时和怎样使用这些技术、方法和支持工具。将基本测试技术和方法制度化有两个子目标：①在组织范围内成立测试技术组，研究、评价和推荐基本的测试技术和测试方法，推荐支持这些技术与方法的基本工具；②制定管理方针以保证在整个组织范围内一致使用所推荐的技术和方法。

3. 集成级

在集成级，测试不仅仅是跟随在编码阶段之后的一个阶段，它已被扩展成与软件生命周期融为一体的一组已定义的活动。测试活动遵循软件生命周期的V字模型。测试人员在需求分析阶段便开始着手制定测试计划，并根据用户或客户需求建立测试目标，同时设计测试用例并制定测试通过准则。在集成级，应成立软件测试组织，提供测试技术，关键的测试活动应有相应的测试工具予以支持。在该测试成熟度等级上，没有正式的评审程序，没有建立质量过程和产品属性的测试度量。在集成级，要实现4个成熟度目标：建立软件测试组织，制定培训计划，开展软件全生命周期测试，控制和监视测试过程。

1）建立软件测试组织

软件测试的过程及质量对软件产品质量有直接影响。由于测试往往是在时间紧、压力大的情况下所完成的一系列复杂的活动，因此应由训练有素的专业人员组成测试组。测试组要完成与测试有关的多种活动，包括负责制定测试计划，实施测试执行，记录测试结果，制定与测试有关的标准和测试度量，建立测试数据库，测试重用，测试跟踪以及测试评价等。建立软件测试组织要实现4个子目标：①建立整个组织范围内的测试组，并得到上级管理层的领导和各方面的支持，包括经费支持；②定义测试组的作用和职责；③由训练有素的人员组成测试组；④建立与用户或客户的联系，收集他们对测试的需求和建议。

2）制定培训计划

为高效率地完成好测试工作，测试人员必须经过适当的培训。制定技术培训计划有3个子目标：①制定组织的培训计划，并在管理上提供包括经费在内的支持；②制定培训目标和具体的培训计划；③成立培训组，配备相应的工具、设备和教材。

3）开展软件全生命周期测试

提高测试成熟度和改善软件产品质量都要求将测试工作与软件生命周期中的各个阶段联系起来。该目标有4个子目标：①将测试阶段划分为子阶段，并与软件生命周期的各阶段相联系；②基于已定义的测试子阶段，采用软件生命周期V模型；③制定与测试相关的工作产品的标准；④建立测试人员与开发人员共同工作的机制。这种机制有利于促进将测试活动集成于软件生命周期中。

4）控制和监视测试过程

为控制和监视测试过程，软件组织需采取相应措施。例如，制定测试产品的标准，制定与测试相关的偶发事件的处理预案，确定测试里程碑，确定评估测试效率的度量，建立测试日志等。控制和监视测试过程有3个子目标：①制定控制和监视测试过程的机制和政策；②定义、记录并分配一组与测试过程相关的基本测量；③开发、记录并文档化一组纠偏措施和偶发事

件处理预案，以备实际测试严重偏离计划时使用。

在 TMM 的定义级，测试过程中引入计划能力，在 TMM 的集成级，测试过程引入控制和监视活动。两者均为测试过程提供了可见性，为测试过程持续进行提供保证。

4. 管理和测量级

在管理和测量级，测试活动除测试被测程序外，还包括软件生命周期中各个阶段的评审、审查和追查，使测试活动涵盖了软件验证和软件确认活动。根据管理和测量级的要求，软件工作产品以及与测试相关的工作产品，如测试计划、测试设计和测试步骤都要经过评审。因为测试是一个可以量化并度量的过程。为了测量测试过程，测试人员应建立测试数据库，收集和记录各软件工程项目中使用的测试用例，记录缺陷并按缺陷的严重程度划分等级；此外，所建立的测试规程应能够支持软件组织对测试过程的控制和测量。管理和测量级有 3 个要实现的成熟度目标：建立组织范围内的评审程序，建立测试过程的测量程序和软件质量评价。

1）建立组织范围内的评审程序

软件组织应在软件生命周期的各阶段实施评审，以便尽早有效地识别、分类和消除软件中的缺陷。建立评审程序有 4 个子目标：①管理层要制定评审政策支持评审过程；②测试组和软件质量保证组要确定并文档化整个软件生命周期中的评审目标、评审计划、评审步骤以及评审记录机制；③评审项由上层组织指定；④通过培训参加评审的人员，使他们理解和遵循相关的评审政策、评审步骤。

2）建立测试过程的测量程序

测试过程的测量程序是评价测试过程质量、改进测试过程的基础，对监视和控制测试过程至关重要。测量包括测试进展、测试费用、软件错误和缺陷数据以及产品测量等。建立测试测量程序有 3 个子目标：①定义组织范围内的测试过程测量政策和目标；②制定测试过程测量计划，测量计划中应给出收集、分析和应用测量数据的方法；③应用测量结果制定测试过程改进计划。

3）软件质量评价

软件质量评价内容包括定义可测量的软件质量属性，定义评价软件工作产品的质量目标等项工作。软件质量评价有 2 个子目标：①管理层、测试组和软件质量保证组要制定与质量有关的政策、质量目标和软件产品质量属性；②测试过程应是结构化、已测量和已评价的，以保证达到质量目标。

5. 优化、预防缺陷和质量控制级

由于本级的测试过程是可重复、已定义、已管理和已测量的，因此软件组织能够优化调整和持续改进测试过程。测试过程的管理为持续改进产品质量和过程质量提供指导，并提供必要的基础设施。优化、预防缺陷和质量控制级有 3 个要实现的成熟度目标：应用过程数据预防缺陷，质量控制，优化测试过程。

1）应用过程数据预防缺陷

这时的软件组织能够记录软件缺陷，分析缺陷模式，识别错误根源，制定防止缺陷再次发生的计划，提供跟踪这些活动的办法，并将这些活动贯穿于整个组织的各个项目中。应用过程数据预防缺陷有 4 个成熟度子目标：①成立缺陷预防组；②识别和记录在软件生命周期各阶段引入的软件缺陷和消除的缺陷；③建立缺陷原因分析机制，确定缺陷原因；④管理人员、开发人员和测试人员互相配合制定缺陷预防计划，防止已识别的缺陷再次发生。缺陷预防计划要具有可跟踪性。

2）质量控制

在本级，软件组织通过采用统计采样技术，测量组织的自信度，测量用户对组织的信赖度

以及设定软件可靠性目标来推进测试过程。为了加强软件质量控制，测试组和质量保证组要有负责质量的人员参加，他们应掌握能减少软件缺陷和改进软件质量的技术和工具。支持统计质量控制的子目标有：①软件测试组和软件质量保证组建立软件产品的质量目标，如产品的缺陷密度、组织的自信度以及可信赖度等；②测试管理者要将这些质量目标纳入测试计划中；③培训测试组学习和使用统计学方法；④收集用户需求以建立使用模型。

3）优化测试过程

在测试成熟度的最高级，已能够量化测试过程。这样就可以依据量化结果来调整测试过程，不断提高测试过程能力，并且软件组织具有支持这种能力持续增长的基础设施。基础设施包括政策、标准、培训、设备、工具以及组织结构等。优化测试过程包含：①识别需要改进的测试括动；②实施改进；③跟踪改进进程；④不断评估所采用的与测试相关的新工具和新方法；⑤支持技术更新。

测试过程优化所需的子成熟度目标包括：①建立测试过程改进组，监视测试过程并识别其需要改进的部分；②建立适当的机制以评估改进测试过程能力和测试成熟度的新工具和新技术；③持续评估测试过程的有效性，确定测试终止准则。终止测试的准则要与质量目标相联系。

8.1.5 软件测试过程改进

软件测试过程也就是软件测试生命周期，它严重影响着软件开发的效率和软件产品的质量。测试技术解决了测试采用的方法和技术问题，测试管理保证了各项测试活动的顺利开展。软件测试过程改进主要着眼于合理调整各项测试活动的时序关系，优化各项测试活动的资源配置以及实现各项测试活动效果的最优化。在软件测试过程中，过程改进被赋予了举足轻重的地位，在测试计划、实施、检查、改进的循环中，过程改进既是一次质量活动的终点，又是下次质量活动的原点，起着承上启下的作用，因此软件测试过程改进对于软件质量的提高相当重要。

1. 软件测试过程改进的概念

测试过程的改进对象应该包括三个方面：组织、技术和人员。需要对组织给予特别关注，因为过程都是基于特定的组织架构建设的，而且组织设置是否合理对过程的好坏有决定性的影响。

1）软件测试组织架构问题

软件测试组织的不良架构通常表现在：①没有恰当的角色追踪项目进展；②没有恰当的角色进行缺陷控制、变更和版本追踪；③项目在测试阶段效率低下、过程混乱；④只有测试经理了解项目，项目成了个人的项目，而不是组织的项目；⑤关心进度，而忘记了项目的另外两个要素——质量和成本。

上述问题可从组织上找出原因。因此在测试过程改进中可以先将测试从开发活动中分离出来，把缺陷控制、版本管理和变更管理从项目管理中分离出来。此外，需要给测试经理赋予明确的职责和目标。技术的改进包括对流程、方法和工具的改进，它包括组织或者项目对流程进行明确的定义，杜绝随机过程，引入统一的管理方法，并使用标准的经过组织认可的工具和模板。人员的改进主要是指对企业文化的改进，它将促使建立高效率的团队和组织。

2）软件测试过程改进战略

由于测试过程改进是一项长期的、没有终点的活动，而且要获得改进过程的收益也是长期的过程，所以在起步实施测试过程改进时，要充分考虑战略，并根据公司的战略目标确定测试

部门的战略，描绘远景。将测试过程改进与公司战略目标相联系，是改进成功实施的必要条件，也是各公司在实施测试过程改进中获得的最佳实践。在研究过程中，组织的规划通常包括如下内容。

(1) 绘制远景：如提升管理成熟度，提高测试生产率，促使部门测试能力达到公司领先水平。

(2) 战略分析：如在部门内制定三年计划。以内部人员为主，引入适当的培训，通过一年半到两年的内部过程，使 V&V 及其他相关过程得到改进并达到 CMMI3 成熟度，适时进行评估，最终目标为 CMMI4。

(3) 优缺点评估：上述战略方法的优点在于前期以内部改进为宗旨，避免了拔苗助长带来的风险，可以使过程改进更符合组织的实际情况。缺点是不以正式评估作为目标，可能导致领导关注力度减弱，过程改进的动力不足，因此需要过程改进的负责人具有坚韧的斗志和持之以恒的信念。

3) 软件测试过程改进策略选择举例

在改进的不同时期和阶段，选择的策略也不同，组织应根据实际情况进行选择，下面列举在研究过程中收集的可供参照的主要策略方法。

(1) 重诊断，轻评估。要以诊断和解决测试过程中的实际问题作为测试过程改进的目的，不能盲目追求商业评估。在以往实施 ISO9000 的过程中曾发现，组织拿证书的愿望常常会冲淡“过程改进”的真正目的。

(2) 实施制度化的同时，建设企业文化。实施全面制度化的管理是过程改进的有效保障，制度和组织文化总是互相依存，没有良好的文化保障，制度化将困难重重；而没有制度的支撑，文化也将是无根之本。

(3) 引入软件工具。推行配置、自动化测试和缺陷跟踪等工具，将有效地分解事务性工作，可以缓解人力资源不足的困难。常见的过程管理方面的工具包括 Rational 公司的 CleaerCase、ClearQuest，CA 公司的 CCC/Harvest 以及 HP 公司的 QC 及 QTP 等。

(4) 建设管理和工程基础。为了解决基础薄弱的问题，需要在测试过程改进前期为相关部门和员工进行基础管理和基本软件工程的课程培训。

(5) 发动全员参与。全员参与可以分三个层面来理解：①站在高于项目管理的层面；②站在项目管理的层面；③站在开发人员和测试人员的层面。充分调动各方面人员的积极性。

(6) 现有过程的重用。该原则可以充分利用现有过程的合理部分，提高被改进过程的可接受程度和使用价值。

2. 软件测试过程改进的具体方法

过程改进在软件测试过程中占有举足轻重的位置，因此为了更好地保证软件质量，测试过程改进是测试人员经常要做的事情，下面列出了一些软件测试过程改进的具体方法。

(1) 调整测试活动的时序关系。在软件测试过程的测试计划中，不恰当的测试时序会引起误工和测试进度失控。例如，具体到某个工程实践中，有些测试活动是可以并行的，有些测试活动是可以归并完成的，有些测试活动在时间上存在线性关系，等等，所有这些一定要区分清楚并且要做最优化调整，否则会对测试进度产生不必要的影响。

(2) 优化测试活动资源配置。在软件测试过程中，必然会涉及人力、设备、场地、软件环境与经费等资源，那么如何合理地调配各项资源给相关的测试活动是非常值得斟酌的，否则会引起误工和测试进度的失控。在测试资源配置中最常见的人力资源的调配，测试部门如果能深入了解员工的专长与兴趣所在，在进行人员分配时，根据各自的特点进行分配，就能对测试活

动的开展起到事半功倍的效果。

(3) 提高测试计划的指导性。测试计划的指导性就是指测试计划的执行能力。在软件测试过程中,很多时候实际的测试和测试计划是脱节的,或者说很大程度上是没有按照测试计划去执行。测试计划的完成不仅仅是起草测试大纲,而是为了确保测试大纲中计划的内容能真正被执行、真正用于指导测试工作,只有这样,测试活动才能高质量地完成,软件质量才能真正得到保证。

(4) 确立合理的度量模型和标准。在测试过程改进中,测试过程改进小组应根据企业与项目的实际情况制定适合自己公司的质量度量模型和标准,做出符合自己公司发展策略的投入。但是质量度量模型和标准的确立不是马上就可以进行的,而是测试过程改进小组随着测试过程的进行不断实践、不断总结、不断改进的。而质量度量模型和标准一旦确立,很多测试活动就不至于陷入过度测试或测试不够的尴尬状态中,使得测试活动在公司与项目不断发展变化的氛围中保持动态平衡。

(5) 提高覆盖率。覆盖率越高,表明测试的质量越高。覆盖率包括内容覆盖和技术覆盖。内容覆盖指的是起草测试计划、设计测试用例、执行测试用例和跟踪软件缺陷,内容覆盖率越高,就越能避免故障被遗漏的情况;技术覆盖指一项技术指标要尽可能地做到测试技术的覆盖,采用科学的方法来验证某项指标,可以更好地保证产品的质量。

除了上边讲的测试过程改进的具体方法外,我们还应注意以下事项:

(1) 必须注意过程改进是跟公司的发展战略相关的,否则只会对测试过程产生不利的影响。

(2) 测试过程的改进并不意味着必须投入大笔资金。

(3) 在测试过程改进中可以参照 CMM 模型与技术。

8.2 软件测试过程管理

现代软件测试过程管理不是仅锁定在测试阶段,软件测试过程管理在各个阶段的具体内容是不同的,但在每个阶段,测试任务的最终完成都要经过从计划、设计、执行到结果分析、总结等一系列相同步骤,这构成软件测试的一个基本过程。通过软件测试过程管理我们要尽量达到测试成本最小化、测试流程和测试内容完备化、测试手段可行化和测试结果实用化的理想目标。

软件测试是软件工程中的一个子过程,为使软件测试工作系统化、工程化,必须合理地对测试过程进行管理,包括签订第三方独立测试合同、制订测试计划、组织项目人员、建立项目环境、监控项目进展等。软件测试过程主要包括软件测试项目的启动、测试需求分析、测试计划制定、测试用例设计、测试实施和执行、测试结果审查和分析等活动及内容(如图 8-4 所示)。

(1) 测试项目启动。首先要确定项目组长,只要把项目组长确定下来,就可以组建整个测试小组,并可以和开发等部门开展工作。接着参加有关项目计划、分析和设计的会议,获得必要的需求分析、系统设计等文档,以及相关产品/技术知识的培训和转移。

(2) 测试需求分析。测试需求通常是以软件开发需求为基础进行分析,通过对开发需求的细化和分解,形成可测试的内容。因此,测试需求的分析包括五个部分:①明确需求的范围;②明确每一个功能的业务处理过程;③不同的功能点与业务的组合;④挖掘显式需求背后的隐式需求;⑤测试需求应全部覆盖已定义的业务流程,以及功能和非功能方面的需求。

(3) 制定测试计划。确定测试范围、测试策略、测试方法,以及对风险、日程表、资源等进

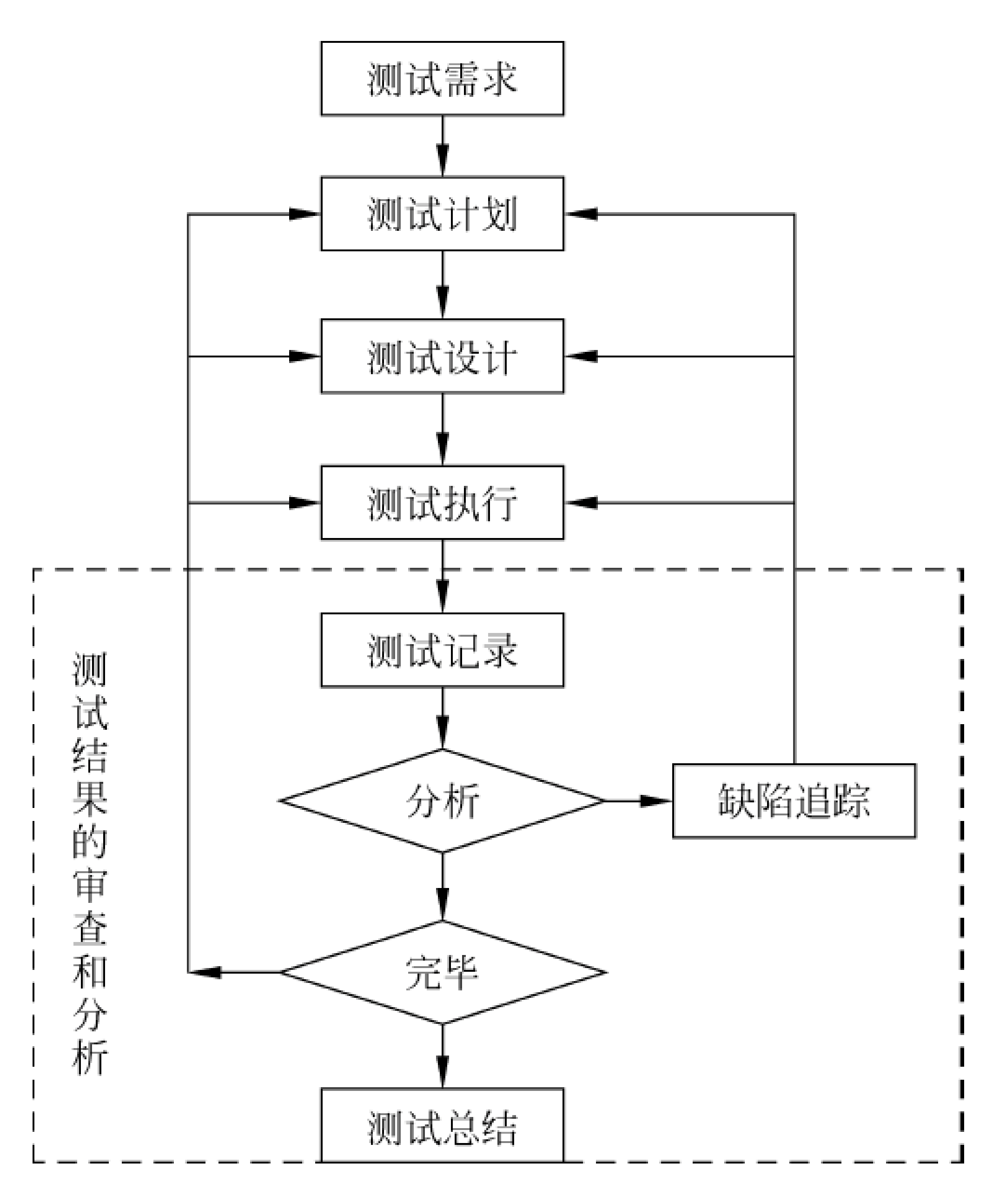


图 8-4 软件测试过程

行分析和估计。

(4) 测试设计和测试开发。制定测试的技术方案、设计测试用例、选择测试工具、编写测试脚本等。测试用例设计要事先做好各项准备,才开始进行,最后还要让其他部门审查测试用例。

(5) 测试实施和执行。建立或设置相关的测试环境,准备测试数据,执行测试用例,对发现的软件缺陷进行报告、分析、跟踪等,测试执行没有很高的技术性,但是测试的基础,直接关系到测试的可靠性、客观性和准确性。

(6) 测试结果的审查和分析。当测试执行结束后,对测试结果要进行整体的或综合分析,以确定软件产品质量的当前状态,为产品的改进或发布提供数据和依据。从管理来讲,要做好测试结果的审查和分析会议,以及做好测试报告或质量报告写作、审查。

根据测试需求、测试计划,对测试过程中每个状态进行记录、跟踪和管理,并提供相关的分析和统计功能,生成和打印各种分析统计报表。通过对详细记录的分析,形成较为完整的软件测试管理文档,避免同样的错误在软件开发过程中再次发生,从而提高软件开发质量。软件测试过程的管理如图 8-5 所示。

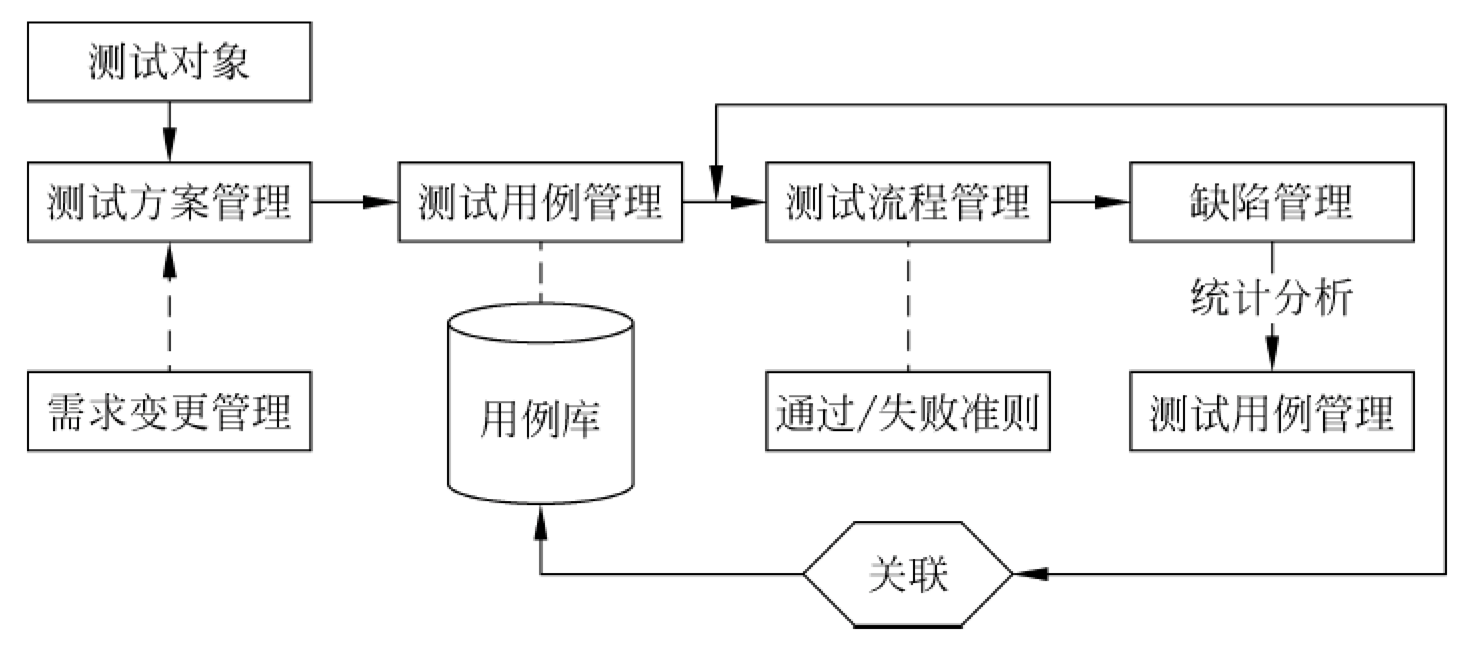


图 8-5 软件测试过程管理

8.2.1 软件测试过程管理的理念

软件测试生命周期模型或软件测试过程模型为我们提供了软件测试的流程和方法，为测试过程管理提供了依据。由于测试过程管理牵涉的范围非常的广泛，包括过程定义、人力资源管理、风险管理等，我们仅从前面介绍的软件测试过程模型（如 W 模型）来介绍软件测试过程管理的思想。

1. 尽早测试

“尽早测试”是从 W 模型中抽象出来的理念。我们说的测试并不是在代码编写完成之后才开展的工作，测试与开发是两个相互依存的并行的过程，测试活动在开发活动的前期已经开展。

“尽早测试”包含两方面的含义：①测试人员早期参与软件项目，及时开展测试的准备工作，包括编写测试计划、制定测试方案以及准备测试用例；②尽早地开展测试执行工作，一旦代码模块完成就应该及时开展单元测试，一旦代码模块被集成为相对独立的子系统，便可以开展集成测试，一旦有系统生成并提交，便可以开展系统测试工作。

由于及早地开展了测试准备工作，测试人员能够于早期了解测试的难度、预测测试的风险，从而有效提高了测试效率，规避测试风险。由于及早地开展测试执行工作，测试人员能够尽早地发现软件缺陷，这样就大大降低了 bug 修复成本。但是需要注意，“尽早测试”并非盲目地提前测试活动，测试活动开展的前提是达到必需的测试就绪点。

2. 全面测试

软件是程序、数据和文档的集合，那么对软件进行测试，就不仅仅是对程序的测试，还应包括软件“副产品”的“全面测试”，这是 W 模型中一个重要的思想。需求文档、设计文档作为软件的阶段性产品，直接影响到软件的质量。阶段产品质量是软件质量的量的积累，不能把握这些阶段产品的质量将导致最终软件质量的不可控。

“全面测试”包含两层含义：①对软件的所有产品进行全面的测试，包括需求、设计文档、代码、用户文档等；②软件开发及测试人员（有时包括用户）全面地参与到测试工作中，例如对需求的验证和确认活动，就需要开发、测试及用户的全面参与，毕竟测试活动并不仅仅是保证软件运行正确，同时还要保证软件满足了用户的需求。

“全面测试”有助于全方位把握软件质量，尽最大可能地排除造成软件质量问题的因素，从而保证软件满足质量需求。

3. 全过程测试

在 W 模型中充分体现的另一个理念就是“全过程测试”。双 V 字过程图形象地表明了软件开发与软件测试的紧密结合，说明了软件开发和测试过程会彼此影响，这就要求测试人员对开发和测试的全过程进行充分的关注。

“全过程测试”包含两层含义：第一，测试人员要充分关注开发过程，对开发过程的各种变化及时做出响应。例如，开发进度的调整可能会引起测试进度及测试策略的调整，需求的变更会影响到测试的执行，等等。第二，测试人员要对测试的全过程进行全程的跟踪，例如建立完善的度量与分析机制，通过对自身过程的度量，及时了解过程信息，调整测试策略。

“全过程测试”有助于及时应对项目变化，降低测试风险。同时对测试过程的度量与分析也有助于把握测试过程，调整测试策略，便于测试过程的改进。

4. 独立的、迭代的测试

我们知道，软件开发瀑布模型只是一种理想状况。为适应不同的需要，人们在软件开发过

程中摸索出了如螺旋、迭代等诸多模型。在这些模型中，需求、设计、编码工作可能重叠并反复进行的，这时的测试工作也将是迭代和反复的。如果不能将测试从开发中抽象出来进行管理，势必使测试管理陷入困境。

软件测试与软件开发是紧密结合的，但并不代表测试是依附于开发的一个过程，测试活动是独立的。只要测试条件成熟，测试准备活动完成，测试的执行活动就可以开展。

所以，我们在遵循尽早测试、全面测试、全过程测试理念的同时，应当将测试过程从开发过程中适当地抽象出来，作为一个独立的过程进行管理。时刻把握独立的、迭代的测试理念，减小因开发模型的繁杂而给测试管理工作带来的不便。对于软件过程中不同阶段的产品和不同的测试类型，只要测试准备工作就绪，就可以及时开展测试工作，把握产品质量。

8.2.2　软件测试计划与测试需求

软件测试计划是指导测试过程的纲领性文件，包括产品概述、测试策略、测试方法、测试区域、测试配置、测试周期、测试资源、测试交流、风险分析等内容。借助软件测试计划，参与测试的项目成员，尤其是测试管理人员，可以明确测试任务和测试方法，保持测试实施过程的顺畅沟通，跟踪和控制测试进度，应对测试过程中的各种变更。测试计划在需求活动一开始就要着手编写，随着开发过程的逐步展开添加内容，在编程活动和单元测试活动之后完成测试计划的编写。测试计划按国家标准或行业标准规定的格式和内容编写。

另外，测试计划最关键的一步就是进行测试需求分析，其测试需求是测试工作的基础。

1. 软件测试计划制定

测试计划要针对测试目的规定测试的任务、所需的各种资源和资金、人员及时间投入、人员角色的安排以及预见可能出现的问题和风险等，以指导测试的执行，最终实现测试的目标，保证软件产品的质量。

1）制定测试计划的目的

编写测试计划的目的是：①为测试各项活动制定一个现实可行的、综合的计划，包括每项测试活动的对象、范围、方法、进度和预期结果；②为项目实施建立一个组织模型，并定义测试项目中每个角色的责任和工作内容；③开发有效的测试模型，能正确地验证正在开发的软件系统；④确定测试所需要的时间和资源，以保证其可获得性、有效性；⑤确立每个测试阶段测试完成以及测试成功的标准、要实现的目标；⑥识别出测试活动中各种风险，并消除可能存在的风险，降低那些不可能消除的风险所带来的损失。

测试计划是一个重要文档，因此在形成测试计划的过程中要对测试计划和测试设计进行检查，当发现错误和遗漏时能在开发过程的早期对测试计划进行必要的增加和修改，减少测试设计的错误。因此形成一份完整的、精确的和全面的测试计划需要经过计划、准备、检查、修改和继续五个步骤。

2）测试计划阶段划分

测试计划不可能一气呵成，而是要经过计划初期、起草、讨论、审查等不同阶段，才能将测试计划制定好。而且，不同的测试阶段（集成测试、系统测试、验收测试等）或不同的测试任务（安全性测试、性能测试、可靠性测试等）都可能要有具体的测试计划。

（1）计划初期是收集整体项目计划、需求分析、功能设计、系统原型、用例报告等文档或信息，理解用户的真正需求，了解技术难点和弱点、或新的技术，和其他项目相关人员交流，在各个主要方面达到一致的理解。

（2）测试计划最关键的一步就是确定测试需求、测试层次。将软件分解成单元，对各个单

元写成测试需求,测试需求也是测试设计和开发测试用例的基础,并用来衡量测试覆盖率的重要指标。

(3) 确定软件测试目标,并使其可以量化、度量和相对集中。可通过对用户需求文档和设计规格文档的分析,来确定被测软件的质量要求和测试需要达到的目标。

(4) 计划起草。根据计划初期所掌握的各种信息、知识,确定测试策略,设计测试方法,完成测试计划的框架。

(5) 内部审查。在提供给其他部门讨论之前,先在测试小组/部门内部进行审查。

(6) 计划讨论和修改。召开有需求分析、设计、开发人员参加的计划讨论会议,测试组长将测试计划设计的思想、策略做较详细的介绍,并听取大家对测试计划中各个部分的意见,进行讨论交流。

(7) 测试计划的多方审查。项目中的每个人都应当参与审查(即市场、开发、支持、技术协作及测试等人员)。计划的审查是必需的,尽管测试工程师努力地完成一个对产品的全面定义,但出自一个测试工程师的定义不一定是完整或准确的。此外,就像开发者很难测试自己的代码那样,测试工程师也很难评估自己的测试计划。每一个计划审查者都可能根据其经验及专长提出修改建议,有时还能提供测试工程师在组织产品定义时不具备的信息。

(8) 测试计划的定稿和批准。在计划讨论、审查的基础上,综合各方面的意见,就可以完成测试计划书,然后报给测试经理或 QA 经理,得到批准,方可执行。测试计划不仅是软件产品当前版本而且还是下一个版本的测试设计的主要信息来源,在进行新版本测试时,可以在原有的软件测试计划书上做修改,但要经过严格审查。

3) 测试计划的要点

软件测试计划的内容主要包括产品基本情况、测试需求说明、测试策略和记录、测试资源配置、计划表、问题跟踪报告、测试计划的评审、结果等。除了产品基本情况、测试需求说明、测试策略等,测试计划的焦点集中在以下方面。

(1) 计划的目的:项目的范围和目标,各阶段的测试范围、技术约束和管理特点。

(2) 项目估算:使用的历史数据,使用的评估技术,工作量、成本、时间估算依据。

(3) 风险计划:测试可能存在的风险分析、识别,以及风险的回避、监控、管理。

(4) 日程:项目工作分解结构,并采用时限图、甘特图等方法制定时间/资源表。

(5) 项目资源:人员、硬件和软件等资源的组织和分配,人力资源是重点,而且日程安排紧密联系。

(6) 跟踪和控制机制:质量保证和控制、变更管理和控制等。测试计划书的内容也可以按集成测试、系统测试、验收测试等阶段去组织,为每一个阶段制定一个计划书,也可以为每个测试任务/目的(安全性测试、性能测试、可靠性测试等)制定特别的计划书。

此外,可对上述测试计划书的每项内容,制定一个具体实施的计划,如将每个阶段的测试重点、范围、所采用的方法、测试用例设计的思想、提交的内容等进行细化,供测试项目组的内部成员使用。对于一些重要的项目,要形成一系列的计划书,如测试范围/风险分析报告、测试标准工作计划、资源和培训计划、风险管理计划、测试实施计划、质量保证计划等。

4) 测试计划的编写内容

我们要按照国家标准或有关行业标准编写测试计划,测试计划要提供被测软件的背景信息、测试目标、测试步骤、测试数据整理以及评估准则。它包括:①测试环境(在不同的条件下进行测试,所得到的结果是不同的。在测试计划中测试环境指用户使用环境或业务运行环境);②测试基本原理和策略;③测试计划阶段划分;④测试计划要点;⑤功能描述和功能覆

盖说明；⑥测试用例清单，说明每个测试用例测什么；⑦测试开始准则和退出准则。

每个测试用例的序言至少包括这些信息：测试用例说明和用途，设置需求（输入、输出），运行测试用例的操作命令，正常和异常信息，编写测试用例的作者。

2. 软件测试需求分析

一个成功的测试项目，首先必须了解测试规模、复杂程度与可能存在的风险，这些都需要通过详细的测试需求来了解。测试需求不明确，只会造成获取的信息不正确，无法对被测软件有一个清晰全面的认识，测试计划就毫无根据可言；另外，测试需求越详细精准，则对被测软件的了解越深，对所要进行的任务内容就越清晰，就更有把握保证测试的质量与进度；最后，软件测试需求是开发测试用例的依据，是衡量测试覆盖率的重要指标。

在进行需求分析时我们要把握3点原则：①制定的测试需求项必须是可核实的。即，它们必须有一个可观察、可评测的结果，无法核实的需求不是测试需求。②测试需求应指明满足需求的正常的前置条件，同时也要指明不满足需求时的出错条件。③测试需求不涉及具体的测试数据，测试数据设计是测试设计环节应解决的内容。

测试需求分析过程实际上就是测试需求的收集、分析与评审过程（如图8-6所示）。

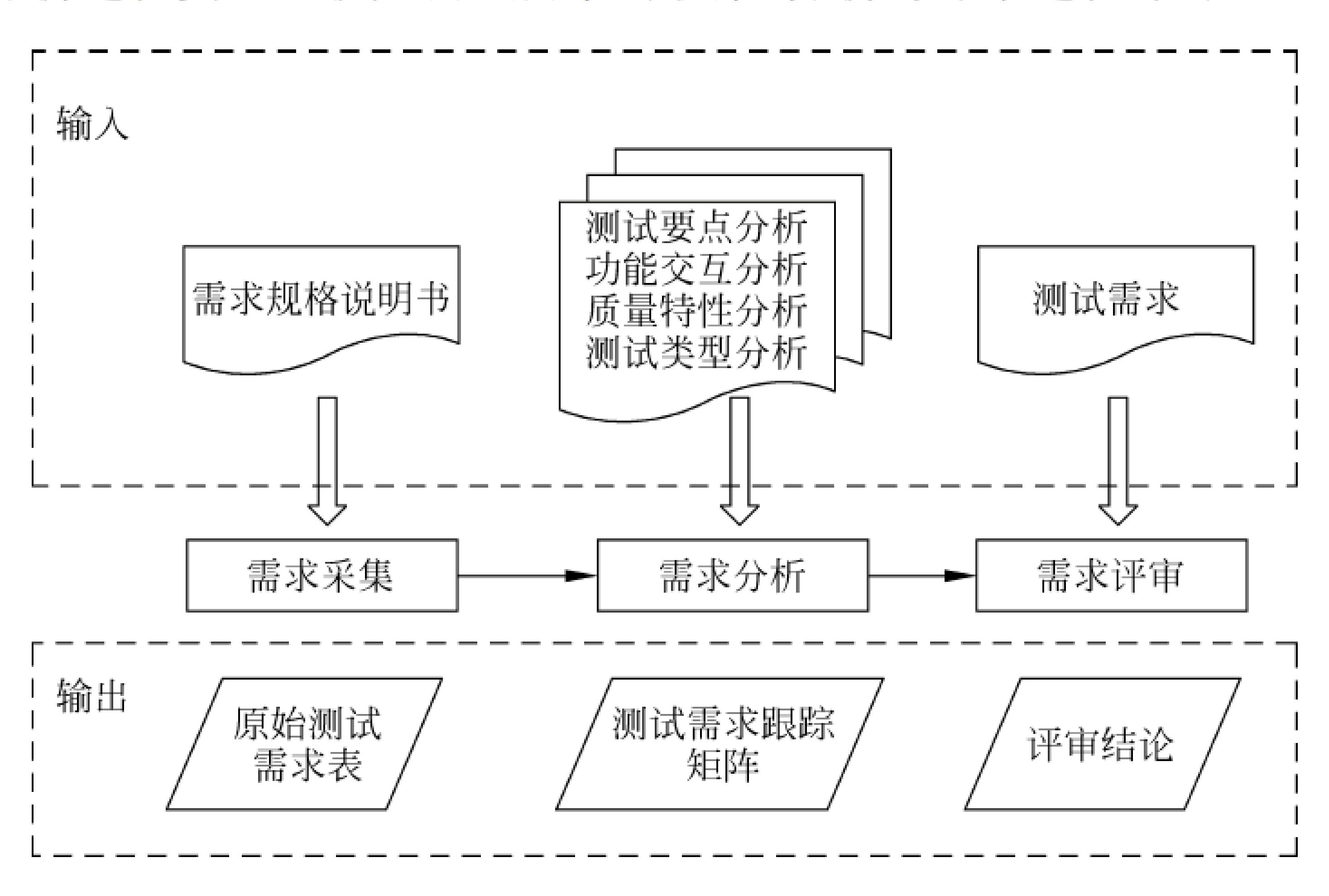


图8-6　软件测试需求分析过程

1）软件测试需求的收集

测试需求通常是以被测对象的软件需求为原型进行分析而转变过来的。但测试需求并不等同于软件需求，它是以测试的观点根据软件需求整理出一个checklist，作为测试该软件的主要工作内容。因此，需求采集的过程实际上是将软件开发需求中的那些具有可测试性的需求或特性提取出来，形成原始测试需求。

所谓可测试性是指这些提取的需求或特性必须存在一个可以明确预知的结果，可以用某种方法对这个明确的结果进行判断、验证，验证是否符合文档中的要求。

测试需求主要通过以下途径来收集。

（1）与被测软件相关的各种文档资料。如软件需求规格、Use case、界面设计、项目会议或与客户沟通时有关于需求信息的会议记录、其他技术文档等。在这个过程中，可通过列表的形式对软件开发需求进行梳理，形成原始测试需求列表（列表的内容包括需求标识、原始测试需求描述、信息来源）。

（2）与客户或系统分析员的沟通。

（3）业务背景资料。如被测软件业务领域的知识等。

(4) 正式与非正式的培训。

(5) 其他。如果以旧系统为原型,以全新的架构方式来设计或完善软件,那么旧系统的原有功能与特性就成了最有效的测试需求收集途径。

在整个信息收集过程中,务必确保软件的功能与特性被正确理解。这要求测试需求分析人员必须具备优秀的沟通能力与表达能力。

2) 软件测试需求的分析

首先,测试需求分析需要考虑几个层面的因素。

第一层:测试阶段

系统测试阶段,需求分析更注重于技术层面,即软件是否实现了具备的功能。如果某一种流程或者某一角色能够执行一项功能,那么我们相信具备相同特征的业务或角色都能够执行该功能。为了避免测试执行的冗余,可不再重复测试。而在验收测试阶段,更注重于不同角色在同一功能上能否走通要求的业务流程。因此需要根据不同的业务需要而测试相同的功能,以确保系统上线后不会有意外发生。但是否有必要进行这种大量的重复性质的测试,要看测试管理者对测试策略与风险的平衡能力了。

目前,大多数的测试都会在系统测试中完成,验收测试只是对于系统测试的回归。此时关键要看测试周期与资源是否允许,以及各测试阶段的任务划分。

第二层:被测软件的特性

不同的软件业务背景不同,所要求的特性也不相同,测试的侧重点自然也不相同。除了需要确保要求实现的功能正确,银行/财务软件更强调数据的精确性,网站强调服务器所能承受的压力,ERP 强调业务流程,驱动程序强调软硬件的兼容性。在做测试分析时需要根据软件的特性来选取测试类型,并将其列入测试需求当中。

第三层:测试的焦点

测试的焦点是指根据所测的功能点进行分析、分解,从而得出的着重于某一方面的测试,如界面、业务流、模块化、数据、输入域等。目前关于各个焦点的测试也有不少的指南,那些已经是很好的测试需求参考了,在此仅列出业务流的测试分析方法。

3) 软件测试需求分析举例

任何一套软件都会有一定的业务流,也就是用户用该软件来实现自己实际业务的一个流程。简单来说,在做测试需求分析时需要列出:常用的或规定的业务流程、各业务流程分支的遍历、明确规定不可使用的业务流程、没有明确规定但是应该不可以执行的业务流程、其他异常或不符合规定的操作等类别。

然后根据软件需求理出业务的常规逻辑,按照以上类别提出的思路,一项一项列出各种可能的测试场景,同时借助于软件的需求以及其他信息,来确定该场景应该导致的结果,便形成了软件业务流的基本测试需求。

在做完以上步骤之后,将业务流中涉及的各种结果以及中间流程分支回顾一遍,确定是否还有其他场景可能导致这些结果,以及各中间流程之间的交互可能产生的新的流程,从而进一步补充与完善测试需求。

另外,在测试需求分析过程中,要确定测试需求的优先级别。这有利于测试工作有的放矢地展开,使测试人员清晰了解核心的功能、特性与流程有哪些,客户最为关注的是什么。由此可确定测试的工作重点在何处,方便处理在测试进度发生问题时,实现不同优先级别的功能、模块、系统等测试,从而缓和测试风险。

通常,需求管理规范的客户,会规定用户需求/软件需求的优先级别,测试需求的优先级可

根据其直接定义。如果没有规定项目需求的优先级，则可与客户沟通，确定哪些功能或特性是需要尤其关注的，从而确定测试需求的优先级。

最后，测试需求分析对测试需求的覆盖率是有要求的。测试需求的覆盖率通常是由与软件需求所建立的对应关系来确定的。如果一个软件的需求已经跟测试需求存在着一对一或一对多的对应关系，可以说测试需求已经覆盖了该功能点，以此类推，如果确定了所有的软件需求都建立了对应的测试需求，那么测试需求的覆盖率便是测试需求覆盖点/软件需求功能点＝100％，但并不意味着测试需求的覆盖程度高。因为测试需求的覆盖率只计算了显性的(即被明确规定的功能与特性)因素，而隐性的(即没有被明确规定但是有可能或不应该拥有的功能与特性)因素并未计算在内。因此根据不断的完善或实际测试中发生的缺陷，可以对测试需求进行补充或优化，并更新添加到测试用例中，以此来提高测试需求的覆盖程度。

4) 软件测试需求的评审

软件测试需求的评审包括如下内容。

(1) 完整性审查：应保证测试需求能充分覆盖软件需求的各种特征，重点关注功能要求、数据定义、接口定义、性能要求、安全性要求、可靠性要求、系统约束等方面，同时还应关注是否覆盖开发人员遗漏的、系统隐含的需求。

(2) 准确性审查：应保证所描述的内容能够得到相关各方的一致理解，各项测试需求之间没有矛盾和冲突，各项测试需求在详尽程度上保持一致，每一项测试需求都可以作为测试用例设计的依据。

一般采用如下形式。

(1) 相互评审、交叉评审：甲和乙在一个项目组，处在一个领域，但工作内容不同，甲的工作成果交给乙审查，乙的工作成果交给甲审查。相互评审是最不正式的一种评审形式，但应用方便、有效。

(2) 轮查：又称分配审查方法，是一种异步评审方式。作者将需要评审的内容发送给各位评审员，并收集他们的反馈意见。

(3) 走查：作者将测试需求在现场向一组同事介绍，以收集大家的意见。希望参与评审的其他同事可以发现其中的错误，并能进行现场讨论。这种形式介于正式和非正式之间。

(4) 小组评审：通过正式的小组会议完成评审工作，是有计划的和结构化的评审方式。评审定义了评审会议中的各种角色和相应的责任，所有参与者在评审会议的前几天就拿到了评审材料，并对该材料进行了独立研究。

(5) 审查：审查和小组评审很相似，但更为严格，是最系统化、最严密的评审形式，包含了制定计划、准备和组织会议、跟踪和分析审查结果等。

评审由以下人员组成：

(1) 正式评审小组中，一般存在多种角色，包括协调人、作者、评审员等。

(2) 评审员需要精心挑选，保证不同类型的人员都要参与进来，通常包括开发经理、项目经理、测试经理、系统分析人员、相关开发人员和测试人员等。

测试需求分析结束后，还有一件很重要的工作，就是制定测试策略。测试策略描述当前测试的目标和所采用的测试方法。其中，当前测试的目标是针对某个应用软件系统或程序的。具体的测试任务目标是：测试要达到的预期结果有哪些，在规定的时间内哪些测试内容要完成，软件产品的特性或质量在哪些方面得到确认等。测试策略还要描述测试不同阶段(单元测试、集成测试、系统测试)的测试对象、范围和方法以及每个阶段内所要进行的测试类型(功能测试、性能测试、压力测试等)。在制定测试策略前，要确定测试策略项，测试策略包括以下几个方面。

(1) 要使用的测试技术和工具,如60%用工具自动测试,40%手工测试。

(2) 测试完成标准,用以计划和实施测试,及通报测试结果。如95%测试用例通过并且重要级别的缺陷全部解决。

(3) 影响资源分配的特殊考虑,如有些测试必须在周末进行,有些测试必须通过远程环境执行,有些测试需考虑与外部接口或硬件接口。

在确认测试方法时,要根据实际情况,结合测试策略的特点来选择合适的方法:

(1) 根据是否需要执行被测软件来划分,有静态测试和动态测试。静态测试,如规格说明书、程序代码的审查,在工作中容易被忽视,在测试策略上应说明如何加强这些环节。

(2) 根据是否针对系统的内部结构和具体实现算法来划分,有"白盒"测试和"黑盒"测试。如何将"白盒"测试和"黑盒"测试有机地结合起来测试,也是测试策略要处理的问题之一。尽管用户更倾向于基于程序规格说明的功能测试,但是"白盒"测试能发现潜在的逻辑错误,而这种错误往往是功能测试发现不了的。

综上所述,可能要在"基于测试技术的测试策略"和"基于测试方案的综合测试策略"之间做出一个选择。

8.2.3 软件测试设计和开发

当测试计划完成之后,测试过程就要进入软件测试设计和开发阶段。软件测试设计是建立在测试计划书的基础上,认真理解测试计划的测试大纲、测试内容及测试的通过准则,通过测试用例来完成测试内容与程序逻辑的转换,作为测试实施的依据,以实现所确定的测试目标。软件设计是将软件需求转换成为软件表示的过程,主要描绘出系统结构、详细处理过程和数据库模式;软件测试设计则是将测试需求转换成测试用例的过程,它要描述测试环境、测试执行的范围、层次和用户的使用场景以及测试输入和预期的测试输出等。所以软件测试设计和开发是软件测试过程中一个技术深、要求高的关键阶段。

1. 测试设计与开发的主要内容

软件测试设计和开发主要内容如下:

(1) 制定测试的技术方案,确认各个测试阶段要采用的测试技术、测试环境和平台,以及选择什么样的测试工具。系统测试中的安全性、可靠性、稳定性、有效性等的测试技术方案是这部分工作内容的重点。

(2) 设计测试用例,根据产品需求分析、系统技术设计等规格说明书,在测试的技术方案基础上,设计具体的测试用例。

(3) 设计测试用例特定的集合,满足一些特定的测试目的和任务,即根据测试目标、测试用例的特性和属性(优先级、层次、模块等),来选择不同的测试用例,构成执行某个特定测试任务的测试用例集合(组),如基本测试用例组、异常测试用例组、性能测试用例组、完全测试用例组等。

(4) 测试开发:根据所选择的测试工具,将所有可以进行自动化测试的测试用例转换为测试脚本的过程。

(5) 测试环境的设计,根据所选择的测试平台以及测试用例所要求的特定环境,进行服务器、网络等测试环境的设计。

软件测试设计中,要考虑的要点有:①所设计的测试技术方案是否可行、是否有效、是否能达到预期的测试目标;②所设计的测试用例是否完整、边界条件是否考虑、其覆盖率能达到多高;③所设计的测试环境是否和用户的实际使用环境比较接近。

其关键是做好测试设计前的知识传递,将设计/开发人员已经掌握的技术、产品、设计等知

识传递给测试人员；同时，要做好测试用例的审查工作，不仅要通过测试人员的审查，还要通过设计/开发人员的审查。

在软件测试设计和开发阶段，按国家标准 GB/T 9386—2008《计算机软件测试文件编制规范》的要求，我们要编写《测试设计说明》、《测试用例说明》、《测试规程说明》、《测试项传递报告》等文档。

2. 测试用例设计的方法和管理

测试设计工作中的一项重要工作就是测试的准备工作。一般来说，由一位对整个系统设计熟悉的设计人员编写测试大纲，明确测试的内容和测试通过的准则，设计完整合理的测试用例，以便系统实现后进行全面测试。测试准备工作中的重要内容就是设计测试用例。

软件测试用例设计的方法有与"白盒"测试和"黑盒"测试相对应的设计方法。"黑盒"测试的用例设计，采用等价类划分、因果图法、边值分析、用户界面测试、配置测试、安装选项验证等方法，适用于功能测试和验收测试。"白盒"测试的用例设计有以下方法：

(1) 采用逻辑覆盖(包括程序代码的语句覆盖、条件覆盖、分支覆盖)的结构测试用例的设计方法。

(2) 基于程序结构的域测试用例设计方法。"域"是指程序的输入空间，域测试正是在分析输入空间的基础上，完成域的分类、定义和验证，从而对各种不同的域选择适当的测试点(用例)进行测试。

(3) 数据流测试用例设计的方法，是通过程序的控制流，从建立的数据目标状态的序列中发现异常的结构测试方法。

(4) 根据对象状态或等待状态变化来设计测试用例，也是比较常见的方法。

(5) 基于程序错误的变异来设计测试用例，可以有效地发现程序中某些特定的错误。

(6) 基于代数运算符号的测试用例设计方法，受分支问题、二义性问题和大程序问题的困扰，使用较少。

测试用例要经过创建、修改和不断改善的过程，一个测试用例具有以下属性：

(1) 测试用例的优先级次序，优先级越高，被执行的时间越早，执行的频率越多。由最高优先级的测试用例组构成基本验证测试，每次构建软件包时，都要被执行一遍。

(2) 测试用例的目标性，有的测试用例是为主要功能而设计，有的测试用例是为次要功能而设计，有的则为系统的负载而设计，有的则为一些特殊场合而设计。因此，需要根据不同的目标设计不同的测试用例。

(3) 测试用例所属的范围，属于哪一个组件或模块，这种属性被用来管理测试用例。

(4) 测试用例的关联性，测试用例一般和软件产品特性相联系的，多数情况下验证某个产品的功能。这种属性可被用于验证被修改的软件缺陷，或对软件产品紧急补丁包的测试。

(5) 测试用例的阶段性，属单元测试、集成测试、系统测试、验收测试中的某一个阶段。这样对每个阶段，构造一个测试用例的集合被执行，并容易计算出该阶段的测试覆盖率。

(6) 测试用例的状态性，当前是否有效，如果无效，被置于 inactive 状态，不会被运行，只有被激活的(active)测试用例才被运行。

(7) 测试用例的时效性，针对同样功能，可能所用的测试用例不同，是因为不同的产品版本在产品功能、特性等方面的要求不同。

(8) 所有者、日期等特性，测试用例还包括由谁、在什么时间创建的，又由谁、在什么时间修改的。

根据上述特性，再结合测试用例的编号、标题、描述(条件、步骤、期望结果)等，就可以对测

试用例进行基于数据库方式的良好管理。测试用例设计完之后，要经过非正式和正式的审查：非正式的审查一般在QA或测试小组(部门)内部进行，包括同测试组人员互相检查，或让资深人员、测试组长帮助审查；正式的审查一般通过正式文档将已设计好的测试用例分发给相应的系统分析、设计人员和程序员，让他们先通读看一遍，将发现的问题记下来。然后由测试组长或项目经理召开一个测试用例审查会，由测试设计人员先对测试用例的设计思想、方法、思路等进行说明，然后系统分析、设计人员和程序员把问题提出来，测试人员回答，必要时做些讨论。

审查完的测试用例，经修改后，就可以直接用于手工测试或用于测试脚本的开发。

3. 测试开发

根据所选择的测试工具脚本语言，如HP QTP VBScript、IBM Rational SQABasic，编写测试脚本，将所有可以进行自动化测试的测试用例转化为测试脚本。其输入就是基于测试需求的测试用例，输出是测试脚本和与之相对应的期望结果，这种期望结果一般存储在数据库中或特定的格式化文件中。

1）测试开发的步骤

首先要设立测试脚本开发环境，安装测试工具软件，设置管理服务器和具有代理的客户端池，建立项目的共享路径、目录，并能连接到脚本存储库和被测软件等。

然后执行录制测试初始化过程、独立模块过程、导航过程和其他操作过程，结合已经建立的测试用例，将录制的测试脚本进行组织、调试和修改，构造成一个有效的测试脚本体系，并建立外部数据集合。

由于被测系统处在不完善阶段，在运行测试脚本的过程中，容易中断，所以在测试脚本开发时，要处理好这种错误，及时记录当时的状态，又能继续执行下去。处理这个问题，有一些解决办法，如跳转到别的测试过程，调用一个能够清除错误的过程等。

2）测试开发常见的问题

测试开发常见的问题有：①测试开发很乱，与测试需求或测试策略没有对应性；②测试过程不可重用；③测试过程被作为一个编程任务来执行，导致脚本可移植性差。这些问题应该避免，在脚本的结构、模块化、参数传递、基础函数等方面要有好的设计。

8.2.4 软件测试执行

当测试用例的设计和测试脚本的开发完成之后，就开始执行测试。测试的执行有手工测试和自动化测试。手工测试在合适的测试环境上，按照测试用例的条件、步骤要求，准备测试数据，对系统进行操作，比较实际结果和测试用例所描述的期望结果，以确定系统是否正常运行或正常表现；自动化测试是通过测试工具，运行测试脚本，得到测试结果。

在测试执行阶段，测试人员要仔细阅读有关资料，包括规格说明、设计文档、使用说明书及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试用例，做好测试前的准备工作。

在本阶段，测试人员要编写《测试日志》、《测试事件报告》文档。

1. 测试阶段目标的检查

要对每个测试阶段(代码审查、单元测试、集成测试、功能测试、系统测试和验收测试、安装测试等)的结果进行分析，保证每个阶段的测试任务得到执行，达到阶段性目标。

1）代码审查

代码审查不是指编程人员互查，而是指测试人员参与的代码会审。代码会审是由一组人通过阅读、讨论和争议对程序进行静态分析的过程。会审小组由组长、两三名程序设计和测试

人员及程序员组成。会审小组在充分阅读待审程序文本、控制流程图及有关要求、规范等文件基础上，召开代码会审会，程序员逐句讲解程序的逻辑，并展开热烈的讨论甚至争议，以揭示错误的关键所在。实践表明，程序员在讲解过程中能发现许多自己原来没有发现的错误，而讨论和争议则进一步促使了问题的暴露。

2）单元测试

单元测试的目的在于发现各模块内部可能存在的各种差错。

单元测试集中在检查软件设计的最小单位——模块上，通过测试发现实现该模块的实际功能与定义该模块的功能说明不符合的情况，以及编码的错误。由于模块规模小、功能单一、逻辑简单，测试人员有可能通过模块说明书和源程序，清楚地了解该模块的I/O条件和模块的逻辑结构，采用结构测试（“白盒”法）的用例，尽可能达到彻底测试，然后辅之以功能测试（“黑盒”法）的用例，使之对任何合理和不合理的输入都能鉴别和响应。高可靠性的模块是组成可靠系统的坚实基础。

单元测试一般由程序员自己做，但必须提交单元测试用例和测试报告，测试人员要审查单元测试用例和测试报告。

3）集成测试

集成测试是将模块按照设计要求组装起来同时进行测试，主要目标是发现与接口有关的问题。如数据穿过接口时可能丢失；一个模块与另一个模块集成可能相互间造成有害影响；把子功能组合起来可能不产生预期的主功能；个别看起来是可以接受的误差可能积累到不能接受的程度；全程数据结构可能有错误等。

测试人员为发现与外部接口及内部参数传递等方面的问题，要抓住关键模块，关键模块应尽早测试，并将自顶向下、自底向上两种测试策略结合起来，对各个模块严格执行。由于涉及系统不同的模块、不同的层次或不同的部门，容易造成一些漏洞、疏忽，要根据设计文档多提问题、集体审查。

4）功能测试

功能测试的目的是向未来的用户表明系统能够按预定要求的功能那样工作，这时的测试是直接操作完整的软件系统，需要站在用户的角度上，尽量模拟用户使用的各种情景，甚至让用户参与测试。

5）系统测试

系统测试的目的是保证系统在实际的环境中能够稳定、可靠地运行下去，包括恢复性测试、安全性测试、强度测试和性能测试等。系统测试技术要求高，占用资源比较多，所以应充分设计好、准备充分。

6）验收测试

验收测试既可以是非正式的测试，也可以是正式的、有计划的测试。其目的是向未来的用户表明系统能够像预定要求那样工作。一个软件产品可能拥有众多用户，不可能由每个用户验收，此时多采用称为α、β测试的过程。α测试是指软件开发公司组织内部人员模拟各类用户对即将发布的软件产品进行测试，试图发现错误并修正。α测试的关键在于尽可能逼真地模拟实际运行环境和可能的用户操作方式。经过α测试调整的软件产品称为β版本。紧随其后的β测试是指组织公司外部的典型用户试用β版本，并要求用户报告异常情况、提出批评意见，然后再对β版本进行改错和完善。

经过上述的测试过程对软件进行测试后，软件基本满足开发的要求，测试宣告结束，经验收后，将软件提交用户。

2. 测试用例执行的跟踪

测试用例执行直接关系到测试的效率、结果，不仅要做到测试效率高，而且要保证结果正确、准确、完整等，其管理关键是提高测试人员素质和责任心，树立良好的质量文化意识，其次要通过一定的跟踪手段从某些方面保证测试执行的质量。

测试执行的跟踪比较容易，按照测试任务和测试周期，可以得到期望的曲线，然后每天检查测试结果，了解是否按预期进度进行。图 8-7 所示为测试执行情况的跟踪曲线。

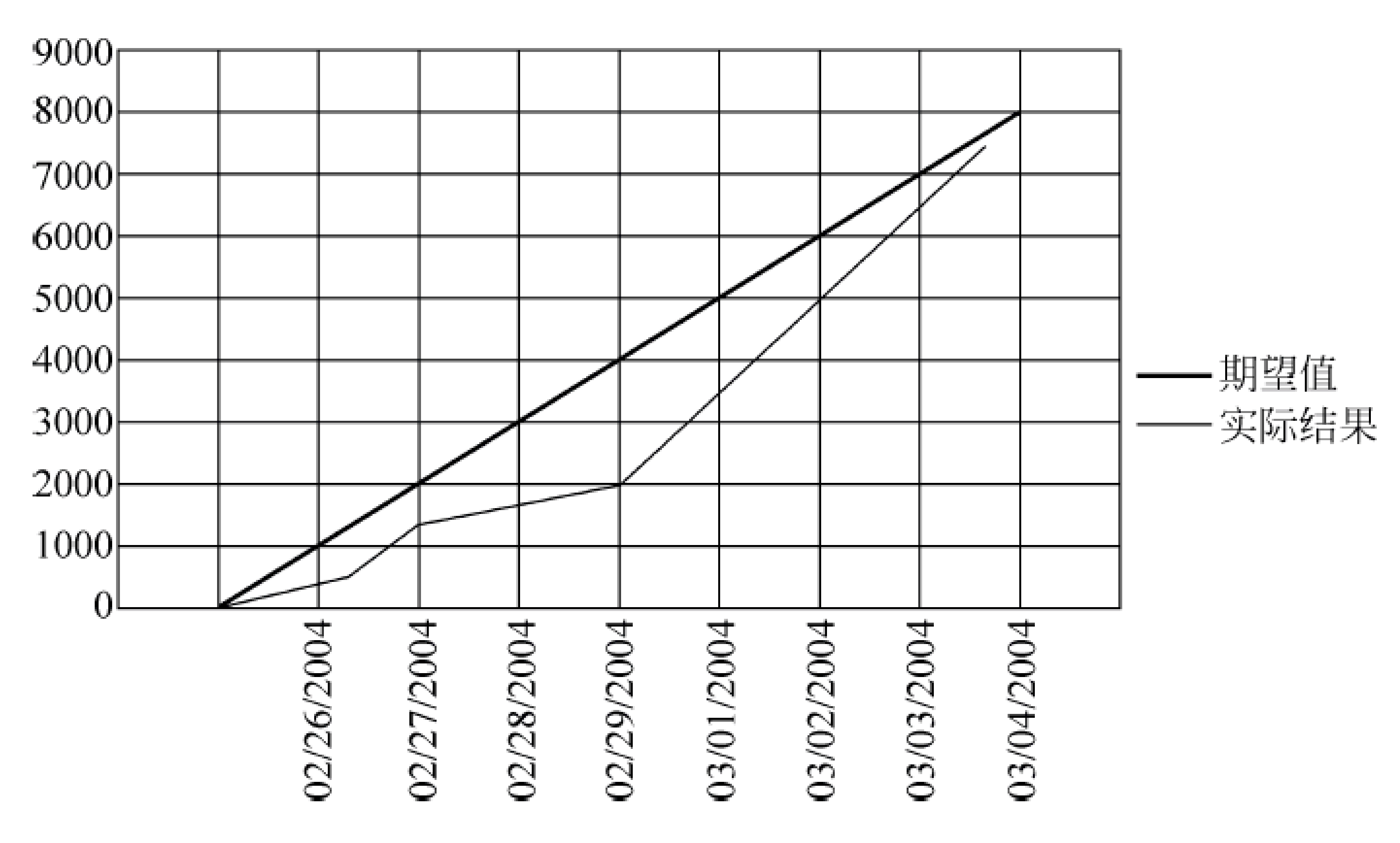


图 8-7 测试执行情况的跟踪曲线

测试结果的跟踪相对存在一些风险，但如果记录好每个人的执行测试情况，即知道哪个测试用例是谁执行的。一旦某个 bug 被漏掉，可以追溯到具体责任人。

3. bug 的跟踪和管理

测试过程中发现的软件错误或缺陷，可提交或纳入到软件缺陷管理过程中。bug 的跟踪和管理一般由数据库系统来执行，但数据库系统也是依赖于一定的规则和流程进行的，主要的思路有：①设计好每个 bug 应该包含的信息条目、状态分类等；②通过系统自动发出邮件给相应的开发人员和测试人员，使得任何 bug 都不会错过，并能得到及时处理；③通过日报、周报等各类项目报告来跟踪目前 bug 状态；④在各个大小里程碑之前，召开有关人员的会议，对 bug 进行会审；⑤通过一些历史曲线、统计曲线等进行分析，预测未来情况。

4. 和项目组外部人员的沟通

为了使测试进展顺利，与项目组外部人员的良好沟通是必要的，这样使测试碰到的问题比较容易解决，测试中发现的 bug 的处理效率也会提高。为此推荐一些方法：①通过一种合适的、可接受的方式指出对方的问题，尽量做到对事不对人；②每周要有一次不同部门参加的会议；③建立大项目的邮件组，包含各部门主要人员的邮件地址；④在同一个大项目组的开发、测试人员的日报、周报等要互相抄送；⑤适当搞些类似于 Party 的活动，改善关系，增加了解。

5. 测试执行结束和测试总结

测试执行全部完成，并不意味着测试项目的结束，测试项目结束的阶段性标志是将测试报告或质量报告发出去后，能得到测试经理或项目经理的认可。除了测试报告或质量报告的写作之外，还要对测试计划、设计和执行等进行检查、分析，完成项目的总结，编写《测试总结报告》。通常包括以下活动。

(1) 审查测试全过程：在原来跟踪的基础上，要对测试项目全过程、全方位地审视一遍，检查测试计划、测试用例是否得到执行，检查测试是否有漏洞。

(2) 对当前状态的审查：包括产品 Bug 和过程中没解决的各类问题。对产品目前存在的

缺陷进行逐个的分析，了解对产品质量影响的程度，从而决定产品的测试能否告一段落。

（3）结束标志：根据上述两项的审查进行评估，如果所有测试内容完成、测试的覆盖率达到要求以及产品质量达到已定义的标准，就可以定稿测试报告，并发送出去。

（4）项目总结：通过对项目中的问题分析，找出流程、技术或管理中存在的根源，避免今后发生，并获得项目成功经验。

8.2.5 软件测试文档

软件测试是一个很复杂的过程，特别是随着测试过程越来越完善，产生的各种测试过程文档也越来越多，这些文档都与相关的测试活动有紧密的联系，它们对于保证软件的质量和它的运行有着重要意义。因此，我们必须把对它们的要求、过程及测试结果以正式的文档形式写出，并将它们放置在组织的软件过程改进库和软件测试配置库中。测试文档的编写是测试工作规范化的一个组成部分，是整个测试过程管理体系具体实施过程的一个重要内容。

1. 软件测试文档概念

软件测试文档描述要执行的软件测试及测试的结果，用来记录、描述、展示测试过程中一系列测试信息的处理过程，通过书面或图示的形式对软件测试过程中的活动或结果进行描述、定义及报告，记载了整个测试的过程和成果。测试文档不只在测试阶段才考虑，它在软件开发的需求分析阶段就开始着手，因为测试文档与用户有着密切的关系。在设计阶段的一些设计方案也应在测试文档中得到反映，以利于设计的检验。测试文档对于测试阶段工作的指导与评价作用更是非常明显的。需要特别指出的是，在已开发的软件投入运行的维护阶段，常常还要进行再测试或回归测试，这时仍须用到测试文档。

作为测试人员，在测试过程中应将各种标准测试文档提交给测试项目组，以确保软件测试项目的质量。也就是说，测试人员的工作绩效与文档的高质量提交也是息息相关的，它描述项目测试过程的每一个细节。因此，从某种程度上说，测试管理的核心内容包括测试文档管理。

从内容上说，软件测试文档大致可以分为测试成果文档和测试过程文档两大类。测试成果文档作为项目可交付物的一个组成部分，其重要性自然不言而喻。测试过程文档主要记录了项目测试过程中的各种信息，为测试人员提供决策依据，以保证项目的顺利实现。另一方面，测试过程文档也是测试过程最为宝贵的资产，通过对过程文档进行归纳和分析，可以对测试项目的成功经验和失败教训了然于胸，从而使后续的测试运作更加有的放矢。

测试项目的阶段性成果是以测试文档形式体现的，测试项目的实施在某种程度上是由测试文档驱动的。从测试文档的角度来看，软件测试过程就是一个文档制作与执行的过程。在软件测试的过程中，每项工作的事前计划、事中测试记录、事后分析结果都要形成相应的测试文档，文档包括与项目相关的资源及其使用情况。

因此，测试文档是软件项目的一部分，没有正式的测试文档的活动，就不是规范的测试。测试文档的编制和管理在软件测试中占有突出的地位和相当大的工作量，高质量地编制、变更、修正、管理和维护文档，对于提高项目测试的质量和客户满意度有着重要的现实意义。

2. 软件测试文档的作用

一个软件测试项目是否高质量完成，一般可以从两个方面进行评价：①能否提供高质量的测试活动和结果；②能否提供有效的测试文档。而对于后者，高质量的测试文档就是前者是否高质量完成的证明。

1）提高软件测试过程的能见度

标准、规范、齐全的文档会详细记录测试过程中发生的事件，便于测试人员掌握测试进度、

测试质量以及各种资源的调配。同时,文档有助于测试人员与开发人员明确了解各自的职责,信息互通,共同把握测试和开发的节奏。

2）文档化能规范测试问题的反馈,提高测试效率

测试人员用一定时间编制、整理测试文档,可以使测试人员对各个阶段的工作都进行周密思考和理顺、找出存在的问题,从而减少差错,提高项目测试质量。例如,测试过程中肯定会遇到各种各样的问题,如软件问题或测试设置等需要向开发组反馈来寻求解决,通过对文档的检查,在项目测试早期发现文档错误和不一致,加以及时纠正,可以减少深入项目而导致的大问题的出现和为纠正失误而付出的更大的成本。

这类问题又分两种情况:①重要的反馈迟迟得不到解决和回复,当文档化做得好时,在出现问题的时候,打开文档可以一目了然,责任没法推卸。②有些问题在不同部门和不同阶段频繁出现。简单而又琐碎的重复问题会让测试人员疲于奔命,效率低下。这时,一个文档化的常见问题集 FAQ 对项目测试就显得意义重大。

3）便于团队成员之间的交流与合作

描述清楚、完备的测试文档便于项目组领导了解测试过程中的各项指标,为开发团队与测试团队之间架起一座桥梁。文档是一种无声的语言,它记录了项目测试过程中有关测试配置、测试运行、测试结果等方面信息,有利于项目管理人员、测试人员之间的交流和合作。

另外,测试文档的重要性还表现在对于项目"传承"的重要性,有了好的文档,当项目有新成员进入,测试文档就可以承担起指导新成员快速工作的作用,而不是单单询问原来的成员,节省了大家的时间。还有,当测试完成后,测试文档就将成为项目测试的文字载体,在后续人员培训方面提供详尽的素材。

4）测试文档是测试人员经验提升的最好途径

善于学习,对于任何职业而言都是前进所必需的动力。对于软件测试来说,这种要求就更加高了,项目文档对于项目测试人员的素质提升大有裨益。目前不少企业在进行项目测试时都会出现一个通病:由于人员素质有限,许多的决定只凭口头叙述,缺少足够的文字记录,以至出现问题时往往显得无所适从。从本质上看,测试文档强调的是一种规范化管理,要求项目人员利用书面语言进行沟通表达,以指引项目运作。

当然,测试人员不应该只为写测试文档而写文档,良好的文档是思想交流、沟通的基础,也是整理和理清思路的基础。不懂得从经验中学习和成长,永远不会有质的提高。只有当每次完成一个测试任务,都有目的的总结,找到自己的不足,一个合格的测试人员才可能成长起来。

5）有利于项目测试的监控作用

测试本身是一项风险很高的工程,需要进行严谨的项目监控。阶段性的检查、评审和文档化成果是重要的方法之一,详尽而规范的测试文档成果不仅有利于监控项目进度,也利于项目验收。

6）有利于测试工作的开展

软件测试文档对测试工作的开展有诸多帮助,例如:

(1) 验证需求的正确性。测试文档中规定了用以验证软件需求的测试条件,研究这些测试条件对弄清用户需求的意图是十分有益的。

(2) 检验测试资源。测试计划不仅要用文件的形式把测试过程规定下来,还应说明测试工作必不可少的资源,进而检验这些资源是否可以得到,即它的可用性如何。如果某个测试计划已经编写出来,但所需资源仍未落实,那就必须及早解决。

(3) 明确任务的风险。有了测试计划,就可以弄清楚测试可以做什么,不能做什么。了解

测试任务的风险有助于对潜在的问题事先做好思想上和物质上的准备。

(4) 生成测试用例。测试用例的好坏决定着测试工作的效率，选择合适的测试用例是作好测试工作的关键。在测试文档编制过程中，按要求精心设计测试用例有重要的意义。

(5) 评价测试结果。测试文档包括测试用例，即若干测试数据及对应的预期测试结果。完成测试后，将测试结果与预期的结果进行比较，便可对已进行的测试提出评价意见。

(6) 再测试。测试文档规定的和说明的内容对维护阶段由于各种原因进行需求再测试时，是非常有用的。

(7) 决定测试的有效性。完成测试后，把测试结果写入文件，这对分析测试的有效性，甚至整个软件的可用性提供了依据。同时还可以证实有关方面的结论。

3. 测试文档常见问题

测试文档是否专业已成为测试管理和测试人员的重要评价指标之一。但是，普遍还会存在以下缺点。

(1) 文档编写不够规范。主要是测试文档内容描述不够完善，在编写各种测试文档过程中，虽然大家都按事先规定的模式进行了编写，但编写的内容经常不够完善。要么文档极其简单，相当于没有文档；要么文档流于形式，没有什么实际的价值，甚至于有的测试文档与测试过程完全不符。

(2) 测试文档没有统一入库管理。随着软件开发的不断深入、升级，新 Bug 不断产生，各种测试文档越来越多，没有建立一个测试文档资料库。在测试过程中没有对每一个阶段的文档进行整理、分层次管理，各类文档资料缺少一致性。不同时期的各种测试文档零散存在，造成查询测试文档时非常困难。在众多的测试文档中，其中一些文档必定是关键文档，起到非常重要的作用，对于这类测试文档没有设定优先级别特别说明。

(3) 只重视测试文档的形式，实用性不强。在实际的测试过程中，编制人员没有时间去关心它们的用途，也不知道哪些部门使用，更多的是在规定的时间内完成任务，以免影响考核成绩。这样一来，一些不实用的、重复的文档不但阻碍着测试的执行效率，而且影响项目的整体进度。因此，文档的制定要实用，以减少繁文缛节的文字工作。

4. 测试文档的管理

从前面我们看到，测试文档对于项目管理的作用是不容置疑的，但测试文档的管理却又通常是项目管理中最容易忽略的。通常在测试文档管理中应该注意以下几个方面。

(1) 建立测试文档管理制度。重点应体现为两点：①要对测试文档的名称、标识、类型、责任人、内容等基本内容做出事先安排，给出测试文档总览表；②制定对各种测试文档的管理程序，如批准、发布、修订、标识、贮存、传递、查阅等，为测试文档配置管理铺设一个良好的基础平台。

(2) 文档版本管理，而且非常重要。版本混乱是测试文档的一个致命伤，测试文档的有效管理必须实行版本控制。

(3) 创建测试文档库的访问规则，这是文档管理的重要环节。访问规则确定谁可以访问、阅读、升级及在文档库中添加文档。同时，文档库还应定期进行检查，以便对哪些文件进行存档或对哪些旧文件进行清理，以确保文档管理符合项目测试组的需求。

(4) 使用工具管理文档。对于一个大型的项目测试，整个测试周期中都会有大量的文档。测试文档内容也是在不断变化的，有的是连续的、承前启后的，有的是新增加的，也有的是废除的。这可能需要一个统一的文档管理工具，分门别类统一存放管理各种测试文档。

总之，测试文档在软件测试过程中起到关键的作用，从某种意义上来说，测试文档是项目

测试规范的体现和指南，按照规范要求编制一整套测试文档的过程，就是完成一个测试项目的过程。

5. 测试文档与测试过程的关系

GB/T 9386—2008《计算机软件测试文档编制规范》对测试文档与测试过程关系有明确的图形表示(见图 8-8)。

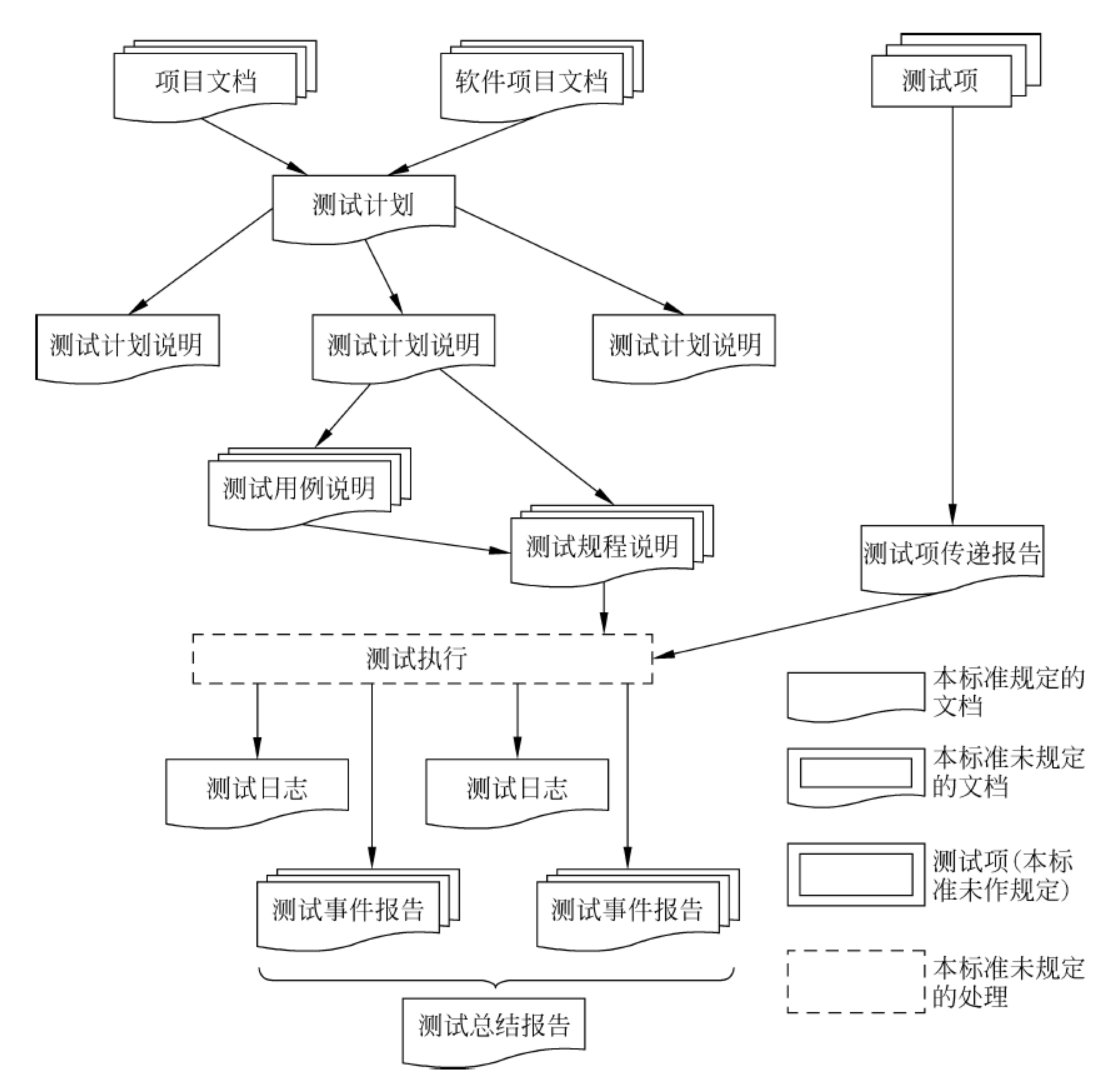


图 8-8 测试文档与测试过程的关系

8.2.6 软件测试用例、测试数据与测试脚本

测试用例是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，以便测试某个程序路径或核实是否满足某个特定需求。它构成了设计和制定测试过程的基础，决定着测试设计和测试开发的类型以及所需的资源。

测试数据是在测试中使用的实际值(集合)或执行测试需要的元素。测试数据创建要测试的条件(作为输入或预先存在的数据)，并且用于核实特定的用例(Use Case)或需求是否已经成功得到实施(将实际结果和预期结果相比较)。

测试脚本是自动执行测试过程(或部分测试过程)的计算机可读指令。测试脚本可以被创建(记录)或使用测试自动化工具自动生成，或用编程语言编程来完成，也可综合前三种方法来完成。测试脚本的目的在于以高效率和有效的方式来实施和执行测试过程。

1. 测试用例

测试用例的设计和编制是软件测试活动中最重要的活动之一。

1）测试用例的作用

测试用例在软件测试中主要有以下作用。

（1）指导测试的实施。测试用例主要适用于集成测试、系统测试和回归测试。在实施测试时测试用例作为测试的标准，测试人员一定要严格按照测试用例所属的项目及其测试步骤逐一实施测试。并将测试情况记录在测试用例管理软件中，以便自动生成测试结果文档。

根据测试用例的测试等级，集成测试应测试哪些用例，系统测试和回归测试又该测试哪些用例，在设计测试用例时都已作明确规定，实施测试时测试人员不能随意作变动。

（2）规划测试数据的准备。通常情况下测试数据是与测试用例分离的。按照测试用例配套准备一组或若干组测试原始数据，以及标准测试结果，即按照测试用例规划准备测试数据是十分必需的。

除正常数据外，还必须根据测试用例设计大量边缘数据和错误数据。

（3）编写测试脚本的"设计规格说明书"。为提高测试效率，软件测试已大力发展自动测试。自动测试的中心任务是编写测试脚本。如果说软件工程中软件编程必须有设计规格说明书，那么测试脚本的设计规格说明书就是测试用例。

（4）评估测试结果的度量基准。完成测试实施后需要对测试结果进行评估，并且编制测试报告。判断软件测试是否完成、衡量测试质量需要一些量化的结果。例如，测试覆盖率是多少、测试合格率是多少、重要测试合格率是多少，等等。以前统计基准是软件模块或功能点，显得过于粗糙。采用测试用例作度量基准更加准确、有效。

（5）分析缺陷的标准。通过收集缺陷，对比测试用例和缺陷数据库，分析确认是漏测还是缺陷复现。漏测反映了测试用例的不完善，应立即补充相应测试用例，最终达到逐步完善软件质量。而缺陷复现表明已有的相应测试用例，在实施测试或变更处理时存在问题。

2）编制测试用例的方法

通常按以下方法编制测试用例。

（1）编写测试用例文档

编写测试用例文档应有文档模板，须符合内部的规范要求。测试用例文档将受制于测试用例管理软件的约束。

软件产品或软件开发项目的测试用例一般以该产品的软件模块或子系统为单位，形成一个测试用例文档，但这并不是绝对的。

测试用例文档由简介和测试用例两部分组成。简介部分编制了测试目的、测试范围、定义术语、参考文档、概述等。测试用例部分逐一列示各测试用例。每个具体测试用例都将包括下列详细信息：测试用例编号、测试用例名称、测试等级、入口准则、测试输入、操作步骤、期望结果（含判断标准）、出口准则、注释等。

（2）测试用例的设置

测试用例可以按功能设置、路径设置以及这两者的组合。

按功能测试是最简捷的，按用例规约遍历测试每一功能。

对于复杂操作的程序模块，其各功能的实施是相互影响、紧密相关、环环相扣的，可以演变出数量繁多的变化。没有严密的逻辑分析，产生遗漏是在所难免的。路径分析是一个很好的方法，其最大的优点是可以避免遗漏测试。

但路径分析法也有局限性。若一个子系统有十余个或更多的模块，这些模块相互有关联。再采用路径分析法，其路径数量成几何级增长，达5位数或更多，就无法使用了。那么子系统模块间的测试路径或测试用例还是要靠传统方法来解决。这是按功能、路径混合模式设置测

试用例的由来。

(3) 设计测试用例

测试用例可以分为基本事件、备选事件和异常事件。设计基本事件的用例，应该参照用例规约(或设计规格说明书)，根据关联的功能、操作按路径分析法设计测试用例。而对孤立的功能则直接按功能设计测试用例。基本事件的测试用例应包含所有需要实现的需求功能，覆盖率达100%。

设计备选事件和异常事件的测试用例，则要复杂和困难得多。往往在设计编码阶段形成的文档对备选事件和异常事件分析描述不够详尽。而测试本身则要求验证全部非基本事件，并同时尽量发现其中的软件缺陷。

可以采用软件测试常用的基本方法——等价类划分法、边界值分析法、错误推测法、因果图法、逻辑覆盖法等设计测试用例。视软件的不同性质采用不同的方法。如何灵活运用各种基本方法来设计完整的测试用例，并最终实现暴露隐藏的缺陷，全凭测试设计人员的丰富经验和精心设计。

(4) 对测试用例的评审

测试用例是软件测试的准则，但它并不是一经编制完成就成为准则。测试用例在设计编制过程中要组织同级互查。完成编制后应组织专家评审，需获得通过才可以使用。评审委员会可由项目负责人、测试、编程、分析设计等有关人员组成，也可邀请客户代表参加。

(5) 测试用例的修改更新

测试用例在形成文档后也还需要不断完善。主要来自三方面的缘故：①在测试过程中发现设计测试用例时考虑不周，需要完善；②在软件交付使用后反馈的软件缺陷，而缺陷又是因测试用例存在漏洞造成；③软件自身的新增功能以及软件版本的更新，测试用例也必须配套修改更新。

一般小的修改完善可在原测试用例文档上修改，但文档要有更改记录。软件的版本升级更新，测试用例一般也应随之编制升级更新版本。

3) 测试用例管理软件

测试用例管理最好是借助测试用例管理软件。它的主要功能有三个：①能将测试用例文档的关键内容，如编号、名称等自动导入管理数据库，形成与测试用例文档完全对应的记录；②可供测试实施时及时输入测试情况；③最终实现自动生成测试结果文档，包含各测试度量值、测试覆盖表和测试通过或不通过的测试用例清单列表。

有了管理软件，测试人员无论是编写每日的测试工作日志，还是出软件测试报告，都会变得轻而易举。

4) 基于需求的测试用例

软件测试的需求有三个层次，即任务需求、用户需求、功能需求，测试需求分析和测试用例设计参照的是软件需求规格说明书。

测试需求的主要来源是系统需求规格说明书，但有些需求是无法从需求文档中获得，可借助概要设计文档或者详细设计文档获得，或直接从最终的软件产品上获得，测试人员依据这些信息编写测试需求。

实际上，测试用例的设计也就是测试需求细化的过程，可以说，有多细的测试需求，就能设计出多细的测试用例，通常采用等价类划分法划分有效和无效的数据集，采用边界值法找到被测软件的输入数据的边界值数据，在基于需求的测试用例设计中，这两种方法既是基础又是补充，当测试数据量比较大时，通常采用自动化测试工具或正交试验法。测试用例的内容项可依

据具体情况而定，通常包含测试用例编号、测试操作步骤和预期结果等。在软件系统测试过程中，软件需求决定了测试用例设计，而测试用例设计的效果则直接决定了整个软件测试项目的成败，因此测试需求分析和测试用例设计是密不可分的，前者是后者的依据，后者是前者的体现，做好需求到测试用例的转化，才能保证整个测试项目的效果。

在软件系统测试过程中，软件测试需求决定了测试用例设计，而测试用例设计关系到测试用例的运行。应该说，设计出了测试用例后，就需要针对性地选择测试用例运行方式。测试用例的运行一般采用手工运行、编写驱动程序运行、借助自动化工具(如 HP QTP)等方式运行。测试用例设计的优劣直接关系着测试用例运行的工作量，编写脚本自动运行程序是解决此问题的有效方式。编写脚本自动运行程序来驱动测试用例是测试用例运行的趋势，这不仅可以节约第一次测试的工作量，而且还可以减少后续的回归测试的工作量。

2. 测试数据

在测试过程中，我们都知道需要做测试用例的设计，但其中很重要的工作是测试数据的设计。因为在测试设计活动中，需要确定和描述两个重要的工作产品：测试过程和测试用例。如果没有测试数据，这两个工作产品将无法实施和执行。它们只是对条件、场景和路径的一些说明，而没有具体的值用来简明地确定它们。测试数据虽然本身不是一个产出物，但是它对测试的成败产生重要的影响。没有测试数据，将无法实施和执行测试，尤其当要求测试数据作为创建条件的输入、作为评估需求的输出结果以及作为支持的值时(测试的前置条件)。因此，确定这些值是一个重要的工作，并且这项工作在确定测试用例后完成。

对测试执行人员来说，测试用例的表示内容包括以下几个方面：测试目标、被测功能点描述、测试运行环境、执行方法(包括测试步骤、输入测试数据或测试脚本)、测试的期望结果、测试的实际结果、其他辅助说明。

从以上几点，我们可以看到输入测试数据只是设计测试用例的一个步骤，而不是全部。

1）测试用例与测试数据的关系

测试用例与测试数据之间的关系是：①测试用例的主体部分包括"测试逻辑"和"测试数据"；②测试用例由主体部分、测试用例相关信息(说明、附件等)和跟踪、管理所需的各种内容组成；③等价类划分、边界值分析等方法主要用于测试数据的设计；④测试逻辑主要包括测试的前提条件、操作步骤和预期结果等；⑤测试逻辑主要通过场景分析来设计。

通过这种方式设计的测试用例，逻辑和数据分离，用例逻辑清晰，内容简洁易理解，也有利于转化成自动化测试脚本。

2）测试数据的属性

确定实际的测试数据时，需说明处理测试数据的四个属性：深度、宽度、范围及构架。

深度是在测试中所使用数据的容量或数量。深度是一个需要考虑的重要事项，因为数据太少可能无法反映现实生活的条件，而数据太多将难以管理和维护。理想条件下，测试应首先使用一个小的支持关键测试用例的数据集(通常为正面测试用例)。随着测试过程中信心不断增强，应该增加测试数据，直到数据深度完全体现出部署环境(或适当可行的范围)为止。

测试数据深度与用做输入和用于支持测试(在预先存在的数据中)的测试数据相关。

宽度是指测试数据值变化的程度。创建更多的记录就可以增加测试数据的宽度和深度。虽然这通常是一个好的解决方法，但是它无法解决我们期望在实际数据中看到的数据真实变化的问题。如果在测试数据中没有这些变化，我们可能无法确定缺陷。

范围是测试数据与测试目标之间的关联关系，它和测试深度和测试宽度相关。具有许多数据并不意味着其数据一定正确。与处理测试数据的宽度一样，我们必须确保测试数据和测

试目标相关。也就是说，需要有支持特定测试目标的测试数据，相当于针对每种场景都要遍历的测试用例相对应的测试数据。

架构是测试数据的物理结构，它仅与测试目标用于支持测试的任何预先存在的数据相关，例如某个应用程序的数据库或规则表。

测试数据的设计是跟随着测试用例的设计而产生的，一般从功能测试的测试数据设计和非功能测试的测试数据设计中产生。

3）测试数据的自动生成

测试数据的自动生成将有效地减轻测试人员的劳动强度，提高测试的效率和质量，节省软件开发的成本。根据估算，对于一个典型的大型软件项目，若能自动生成测试数据，则能节省整个软件开发费用的4%，这是很可观的。HP等公司都有对测试数据自动生成的工具支持。

3. 测试脚本

软件测试的最终目的是让用户满意他们所使用的软件，但要使最终用户对软件感到满意，最有力的举措就是对最终用户的期望加以明确阐述，以便对这些期望进行核实并确认其有效性。测试用例反映了要核实的需求。然而，核实这些需求可能通过不同的方式并由不同的测试员来实施。例如，执行软件以便验证它的功能和性能，这项操作可能由某个测试员采用自动测试技术来实现，由测试脚本支撑。

测试脚本一般指的是一个特定测试的一系列指令，这些指令可以被自动化测试工具执行。为了提高测试脚本的可维护性和可重用性，必须在执行测试脚本之前对它们进行构建。或许会发现这样的情况，即有的操作将出现在几个测试过程中。因此，应有目的地确定这些操作的目标，这样就可以重用它们的实施。

1）测试脚本用途

更改目标软件时，需要对测试过程进行局部的可控制的变更。这将使得测试过程和测试脚本对目标软件的变化有更大的应变能力。例如，假设软件的登录部分已经改变，在遍历该登录部分的所有测试用例中，只有关于登录的测试过程和测试脚本需要进行改变。

测试脚本是针对一个测试过程的。一个测试过程往往需要众多的数据来测试。通过自动录制得到的脚本，所有的输入数据都是常数，是固定的。

如果需要使用一个测试脚本测试多组数据，就需要对脚本进行参数化，把固定的常数修改为来自数据源变量。

2）测试脚本语言

测试脚本语言是脚本语言的一种，准确地讲是脚本语言在测试领域的一个分支，是自动化软件测试设计的基础。

脚本语言工作的核心是脚本解释器，所有具体指令或函数的执行都由它来完成，扩展项实现了与其他语言的接口，使脚本语言运行C/C++、Java等函数成为可能；同时在用户具体应用中可以定义命令和函数，应用更加灵活；作为解释器它也提供了基本的内建指令或是函数，不同厂商、版本的解释器提供的内建命令（函数）可能不同。

8.2.7 软件测试过程中的配置管理

软件测试需要进行充分的测试准备，需要科学的、规范的测试过程管理。有效的配置管理对跟踪和提高测试质量和效率起到十分重要的作用。测试过程中的配置管理工作不仅包括搭建满足要求的测试环境，还包括获取正确的测试、发布版本。

软件测试配置管理一般应用过程方法和系统方法来建立软件测试管理体系，即把软件测

试管理作为一个系统,对组成这个系统的各个过程加以识别和管理,以实现设定的系统目标。同时要使这些过程协同作用、互相促进,从而使它们的总体作用大于各个过程之和。软件测试配置管理的主要目标是在设定的条件限制下,尽可能发现和找出软件缺陷。测试配置管理是软件配置管理的子集,作用于测试的各个阶段。其管理对象包括软件测试计划、测试方案(用例)、测试版本、测试工具及环境、测试结果等。

1. 测试配置管理的目标和阶段

软件测试配置管理的目标是:①在测试过程中控制和审计测试活动的变更;②在测试过程中随着测试项目里程碑的变化,同步建立相应的基线;③在测试过程中记录并且跟踪测试活动过程中的变更请求;④在测试过程中针对相应的软件测试活动或者产品,测试人员应将它们进行标识和控制以使之可用。

软件测试配置管理的阶段划分为:

(1) 需求阶段。我们要进行客户需求调研和软件需求分析。

(2) 设计阶段。在这个阶段我们要进行概要设计和详细设计工作。

(3) 编码阶段。这时我们主要进行的工作是编码。

(4) 测试和试运行阶段。在这个阶段我们要进行单元测试、用户手册编写、集成测试、系统测试、安装培训、试运行和安装运行等工作。

(5) 正式运行及维护阶段。即对产品进行发布和不断的维护。

在软件测试的过程中会产生很多东西,如测试的相关文档和测试各阶段的工作成果。这些包括测试计划文档、测试用例以及自动化测试执行脚本和测试缺陷数据等。为了以后可能的查阅和修改,应该将这些工作成果和文档通过软件测试配置管理保存起来。

2. 测试配置管理过程

通常测试配置管理过程要包括:

(1) 建立测试配置控制委员会。该委员会要做到对项目的每个方面都有所了解。

(2) 软件配置管理(SCM)库的建立和使用。要求在每一个项目过程中都要建立、使用和维护一个软件配置管理库,由配置管理工具支持。这有助于在技术和管理这两个方面对所有的配置项进行控制,并且对它们的发布和有效性也能起到控制作用,同时还能对软件配置管理SCM 库进行备份,以防发生意外或者风险时,能够作为保存灾难恢复备份的副本。

(3) 配置状态报告。配置状态报告是软件测试配置管理过程中的一项重要的活动,在软件测试配置管理过程中配置人员要管理和控制所有提交的产品,然后在有产品提交或者变更结束时,配置人员要进行相应的质量检查。而在这之后,配置人员不但要将批准通过的配置项放入基线库中,并且还要记录配置项及其状态,编写配置状态说明和报告。通过配置人员的这些工作来确保所有应该了解情况的组或者个人能够及时地知道相关的信息。

(4) 评审、审计和发布过程。为了保持 SCM 库中内容的完整性和质量,要求采取适当的质量保证活动来应对 SCM 库中各项的变化。以此来确保在基线发布之前能够执行审计活动。该活动包括基线审计、基线发布和产品构造。

3. 测试配置管理的主要参与人员及其分工

软件测试配置管理中人员的角色分配和分工是确保配置管理活动在软件开发、测试和维护过程中得到贯彻执行的重要保障。因此在制定测试配置管理计划和开展测试配置管理之前,首先要确定配置管理活动的相关人员以及他们的职责和权限。

1) 项目经理

项目经理作为整个软件的开发以及整个软件的维护活动的负责人其主要职责包括采纳软

件测试配置控制委员会的建议,对配置管理的各项活动进行批准,并且在批准之后还要控制它们的进程。项目经理的具体工作职责如下:首先是制定项目的组织结构以及配置管理策略;其次是批准和发布配置管理计划;然后要对项目起始基线和软件开发工作里程碑进行制定;最后是接受并审阅测试配置控制委员会的报告。

2)测试配置控制委员会

测试配置控制委员会的职责是对配置管理的各项具体活动进行指导和控制,并且为项目经理的决策提供建议。该委员会的具体工作职责如下:首先是要批准软件基线的建立以及配置项的标志;其次是制定访问控制策略;然后是建立、更改基线的设置和审核变更申请;最后是根据配置管理员的报告从而决定相应的对策。

3)配置管理员

根据制定的配置管理计划执行各项管理任务,这就是配置管理员的职责,配置管理员要定期向测试配置控制委员会提交报告,同时还要列席他们的例会。配置管理员的具体工作职责如下:①对软件配置管理工具进行日常管理与维护;②提交配置管理计划;③各配置项的管理与维护;④执行版本控制和变更控制方案;⑤完成配置审计并提交报告;⑥对开发人员进行相关的培训;⑦对开发过程中存在的问题加以识别并制定解决方案。

4)开发人员

开发人员的职责为:在了解了项目组织确定的配置管理计划和相关规定之后,按照配置管理工具的使用模型来完成开发任务。

总之,软件测试配置管理要求做到:①每个测试项目的配置管理责任明确;②配置管理贯穿项目的整个测试活动;③配置管理应用于所有的测试配置项,包括支持工具;④建立配置库和基线库;⑤定期评审基线库内容和测试配置项活动。

4. 测试配置管理中的版本控制

配置管理中的一项重要内容就是版本控制。软件测试的版本控制简单地说就是对测试有明确的标识、说明,并且测试版本的交付是在项目管理人员的控制之下。用来识别所用版本的状态就是对测试版本的标识、对不同的版本进行编号,软件质量稳定度趋势的反映也可以由测试的版本控制体现。版本控制是软件测试的一门十分实用的实践性技术,将各次的测试行为以文件的形式进行记录,并且对每次的测试行为进行编号,标识公布过的每一个测试版本,以此来进行测试排序和管理。

版本控制的作用是跟踪、记录整个测试过程,包括测试本身和相关文档,以便对不同阶段的被测软件有相关文档进行标识和差别分析,便于协调和管理测试工作。

为了有效地控制软件测试版本,可采用以下方法。

(1)制定合理的版本发布计划,并加强版本控制管理。软件版本的发布应有计划地执行,在项目开发计划中明确发布版本的时机和版本更新的策略,简单地说就是明确在什么条件下可以发布测试初始版本;什么条件下进行主版本、子版本的更新。这样可以避免频繁地发布测试版本和随意修改版本号给测试带来的混乱。

(2)强化测试准入条件。软件测试的启动是有条件的,在开发人员发布测试版本时,应有相应的文档如自测报告、软件版本说明等支持,这是前提,不满足这个前提,测试活动不应启动。

软件版本说明可以使测试人员了解当前版本的具体内容和与上一版本的差异。

自测报告对版本质量来说是一个重要的保证,它能有效避免出现版本质量太差的情况。

(3) 强化 bug 管理。充分使用测试管理工具,做好 bug 管理工作。首先,在提交 bug 时,应完整填写 bug 的详细内容,如发现版本、对应人员、发现的模块等;其次,在开发修改 bug 时,应注明修复问题的信息;最后,在测试人员关闭 bug 时,应填写选择关闭 bug 的版本号。这有利于分析不同版本和不同模块的 bug 走势。

(4) 做好版本控制的文档管理工作。在每次提交测试版本,执行测试时都会生成相应的文档以便记录版本信息和测试记录及测试结果等,管理好这些文档,不仅有助于跟踪和监测测试版本的执行,而且也便于对测试活动的追溯。

(5) 积极解决问题的态度。无论是开发人员还是测试人员,在版本控制过程中都应有积极的态度,遇到问题及时沟通,以高效的方式来解决问题。

对软件测试的版本控制来说,衡量其效果的标准可归根结底为两点:效率和质量。如果版本控制最终使软件测试效率得到提高、使软件质量得到提升,那就是成功的。反之,则是失败的。

8.2.8 软件测试过程中的组织管理

在工程化的软件项目过程中,软件测试活动贯穿整个软件项目的生存周期。独立的软件测试组织始终与设计、实现、维护组织并行工作。软件测试涉及的人力、物力、时间等资源甚至可能超过软件项目总消耗的一半以上。面对着极其错综复杂的问题,人的主观认识不可能完全符合客观现实,与工程密切相关的各类人员之间的沟通和配合也不可能完美无缺,因此,在软件生命周期的每个阶段都不可避免地会产生差错。为了尽可能少地产生差错,尽可能多地找出程序中的错误,生产出高质量的软件产品,加强对测试工作的组织和管理就显得尤为重要。

1. 软件项目测试组织的建立

由于测试的目标是暴露程序中的错误,从心理学角度看,由程序的编写者自己进行测试是不恰当的。因此,软件项目的测试通常由其他人员组成测试团队或建立测试组织来开展测试工作。好处是:①有利于与软件企业项目管理人员、开发人员及用户进行专业交流;②有利于提高对测试工作、测试管理的重要性的认识,以改进其测试过程,提高测试的质量;③有利于从理论角度、专业角度来认识软件测试和测试管理。

1) 测试部门的组织形式

一个好的组织结构,可以更好地发挥人员的能动性,使工作更有效率,也使工作的质量更高。因此,在组建测试团队之初一定要做好规划:首先可根据测试工作范围和目标估计工作量,确定技术要求;然后确定所需的角色和职责,明确岗位、技术和能力需求;最后生成人员配备管理计划。

常见的测试组织结构可分为以下几类。

(1) 烟囱测试组。烟囱测试组分小型和大型两类,测试人员一般由临时人员组成。小型烟囱测试组,通常由 2~5 人组成,直接向项目经理负责。大型的可以划分为几个小组,设测试经理。项目经理负责制定测试计划文档。问题是企业没有正规的方法将测试程序、方法、相关的知识经验传递下去,测试质量难以保证。优点是成本低,不需要对测试人员提供培训、生活保障等服务。

(2) 集中测试组。企业成立专职、独立的测试部门,通常由 10~30 人组成。集中测试组为每个项目配备几个全职的测试人员,部分企业中可能还负责执行项目中软件质量管理和性

能规范制定的工作,可以将相关的知识、经验传递下去。

(3) 独立验证与确认测试组(IV&V 组)。通常由软件开发组织之外的人员或其中的独立人员组成,如转包商。其参与检查、验证是否遵循标准、进行软件文档的质量保证检测,主要完成系统测试。可以将其看做一个最苛求的用户。

(4) 系统方法与测试组(SMT 组)。通常作为企业的内部顾问组的方式存在。主要负责方法及标准的知识交流、编制开发和测试指南、开发测试方法、测试工具评估与培训,并与不同的项目组进行协作,对其进行指导。通常不负责具体测试工作的执行,由软件专家组成。

图 8-9 与图 8-10 分别展示了两种不同的组织形式。

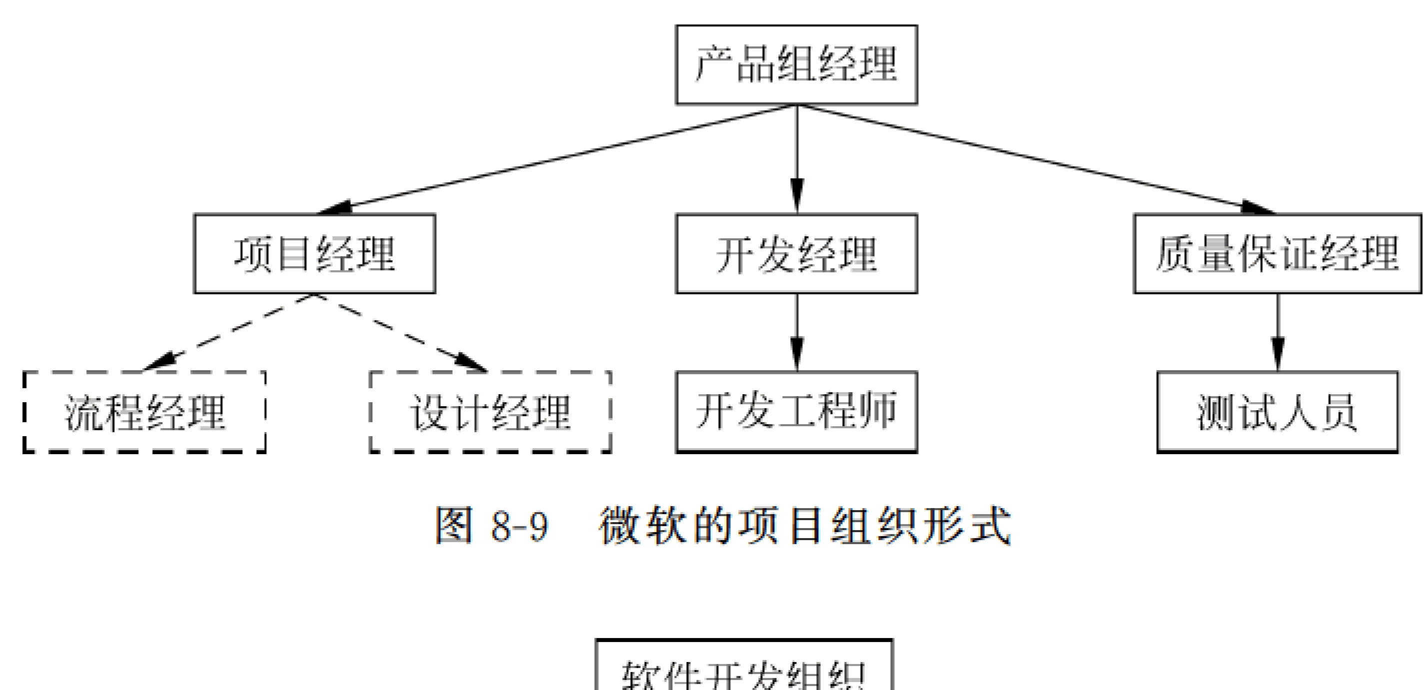


图 8-9 微软的项目组织形式

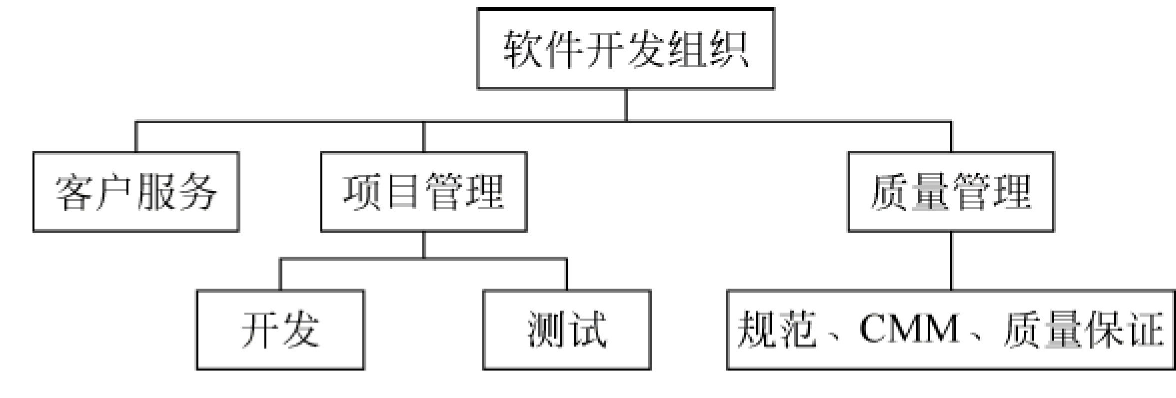


图 8-10 常见的项目组织形式

2) 软件项目测试组织的人员组成

一个成功的测试组必须在 10 个方面满足相关要求:①业务知识(测试工程师应具备业务知识,并和用户紧密接触);②技术知识(熟悉所测试的产品用到的技术,并掌握测试工具、方法等相关技术);③任务划分(将业务任务和技术任务相互独立);④资源管理(业务资源和技术资源相互结合);⑤与开发组的关系(同开发人员协同工作);⑥生命周期早期介入(测试应在开发周期的早期介入);⑦测试过程(有成熟的测试过程管理规范);⑧灵活性/适应性(能够适应不同的测试项目);⑨度量(掌握度量的方法,以改进工作);⑩过程改进(应致力于工作的不断改进)。

一般情况下,测试组的人员组成包括:①测试经理,负责测试流程、沟通、测试工具的引入、人员管理、测试计划/设计/开发及执行;②测试组长,负责沟通、测试工具引入、人员管理、费用/过程状态报告、测试计划/设计/开发及执行;③测试工程师,负责执行测试计划,进行设计/开发及执行。

一个测试项目的测试组规模受到很多因素的影响,如企业文化或测试成熟度、测试需求范围、工程师技能水平、测试工具及应用水平、业务知识、组织形式及测试工作介入时间等。

确定测试组规模有以下几种方法。

(1) 开发比例法:根据开发人员数量按照一定比例来确定测试工程师的数量(表 8-1)。开发人员指进行设计、开发、编译以及进行单元测试的人员。

表 8-1　开发比例法

开 发 类 型	开发人员/人	比例	测试组规模
商业产品(大型市场)	30	3∶2	20
商业产品(小型市场)	30	3∶1	10
单个客户端的应用开发	30	6∶1	5
单个客户端开发并与系统集成	30	4∶1	7
政府部门应用开发(内部)	30	5∶1	6
公司应用开发(内部)	30	4∶1	7

(2) 百分比法：根据测试人员应该占到项目组中人员的百分比数量。百分比法参见表 8-2。

表 8-2　百分比法

开 发 类 型	项目人员/人	比例/%	测试组规模
商业产品(大型市场)	50	27	13
商业产品(小型市场)	50	16	8
单个客户端的应用开发	50	10	5
单个客户端开发并与系统集成	50	14	7
政府部门应用开发(内部)	50	11	5

(3) 测试程序法：根据测试程序数量，以及每个程序可能的执行时间，计算出人小时，再根据完成周期计算测试组规模，如表 8-3 所示。

表 8-3　测试程序法

	测试过程数目	计算因子	人小时	完成周期	测试组规模
历史记录	860	6.16	5300	9 个月	3.4
新项目评估	1120	6.16	6900	12 个月	3.3

(4) 任务计划法：根据历史记录中类似项目工作量，比较新项目同历史项目的工作量，历史项目乘以相应的因子。其步骤是先将任务分解，根据历史记录乘以一个因子，计算出新项目的所有任务工作量，然后再根据该工作量和完成周期计算测试组规模。

测试工作对测试组的人员有很多要求，理想状态下应具备：①适应各种环境的知识背景，学习速度快，组织能力强，解决问题的能力强；②具有创造性，分析和编程能力强；③业务领域的知识深，交流与协调能力强；④具有测试经验，能够关注细节，文字表达、口头表达能力强等。为此，可采取下列培养途径培养测试人员。

(1) 测试职位：初级测试工程师→测试工程师→高级测试工程师→测试组负责人→测试负责人→测试经理→产品/业务经理。

(2) 技术技能：测试工具→测试自动化编程→编程语言→操作系统→网络、数据库→测试生命周期，需 1～2 年。

(3) 测试过程：手工测试，测试用例编写，测试执行，测试文档编写，测试设计，测试计划制定，测试需求分析，评审、制定和改进过程，指导初级工程师工作，了解业务领域，需 3～4 年。

(4) 测试组工作：任务安排、跟踪和报告，监管测试工程师，掌握测试周期支持工具，需 4～6 年。

(5) 项目管理：管理项目，与客户交流，管理测试人员，需 6～12 年。

(6) 产品管理：项目或产品研发指导，促进产品销售，确定业务机会，承担盈亏责任，需12年以上。

2. 测试人员分阶段的投入

为了保证软件的开发质量，软件测试应贯穿于软件定义与开发的整个过程。因此，对分析、设计和实现等各阶段所得到的结果，包括需求规格说明、设计规格说明及源程序都应进行软件测试。基于此，测试人员的组织也应是分阶段的。

1) 需求评审

软件的设计和实现都是基于需求分析规格说明进行的。需求分析规格说明是否完整、正确、清晰是软件开发成败的关键。为了保证需求定义的质量，应对其进行严格的审查。审查小组的人员组成有：组长1人，成员包括系统分析员，软件开发管理者，软件设计、开发和测试人员和用户。

2) 设计评审

软件设计是将软件需求转换成软件表示的过程，主要描绘出系统结构、详细的处理过程和数据库模式。按照需求的规格说明对系统结构的合理性、处理过程的正确性进行评价，同时利用关系数据库的规范化理论对数据库模式进行审查。评审小组的人员组成有：组长1人，成员包括系统分析员、软件设计人员、测试负责人员各1人。

3) 程序的测试

软件测试是整个软件开发过程中交付用户使用前的最后阶段，是软件质量保证的关键。软件测试在软件生命周期中横跨两个阶段：通常在编写出每一个模块之后，就对它进行必要的测试(称为单元测试)。编码与单元测试属于软件生命周期中的同一阶段。该阶段的测试工作，由编程组内部人员进行交叉测试(避免编程人员测试自己的程序)。这一阶段结束后，进入软件生命周期的测试阶段，对软件系统进行各种综合测试。测试工作由专门的测试组完成。测试组设组长1名，负责整个测试的计划、组织工作。测试组的其他成员由具有一定的分析、设计和编程经验的专业人员组成，人数根据具体情况可多可少，一般3～5人为宜。

3. 测试小组(对应小型烟囱测试组)的运行

测试的工作实体(最小组织单位)是测试小组和支持小组，分别由小组长全权负责。小组长向测试主管负责。

测试小组内部的角色分为测试人员、测试工程师和测试支持人员(管理人员、测试环境建设人员属于支持人员)。

测试小组根据测试项目或评测项目的需要临时组建，小组长也是临时指定。与项目组的最大区别是生命周期短，一般是2周到4个月。在系统测试期间或系统评测期间，测试组长是测试对外(主要是项目组)的唯一接口，对内完全负责组员的工作安排、检查和进度管理。

支持小组按照内部相关条例负责测试的后勤保障和日常管理工作，机构设置一般相对比较稳定。主要负责网络管理、数据备份、文档管理、设备管理和维护、员工内部培训、测试理论和技术应用、日常事务管理和检查等。

另外，测试对于每一个重要的产品方向，均设置1～3个人长期研究和跟踪竞争对手的产品特征、性能、优缺点等。在有产品测试时，指导或参加测试(但不一定作为测试组长)，在没有产品测试时，进行产品研究，并负责维护和完善测试设计。目前希望在需求分析阶段多多参与。

测试小组的一般运行方式如下：

(1) 项目组提交系统测试申请给测试部门并指定账号(包括版本控制服务器上的位置)。

由专人检查文档格式和完备性(文档包括需求文档等相关文档)。

(2) 检查合格后由测试主管审查并通过,成立测试组,指定测试组长(可暂时没有组员)。

(3) 测试组长根据该产品的申请报告、测试规范和以往测试数据,制定测试方案和测试计划。

(4) 测试主管审核通过测试计划和测试方案后,根据测试计划指定测试小组成员,并由支持小组完成其他支持任务(如设备的配备、测试数据库的建立、网络权限的修改)。

(5) 测试期间,测试小组根据测试计划进行实际测试,记录并跟踪测试缺陷报告,填写测试记录。测试期间测试组长与项目组(测试经理)经常沟通,并获取产品的更新版本。同时,测试组长审查、修改并提交所有缺陷报告,保证随时掌握产品的质量情况,并监督测试进度。

(6) 产品进行到一定阶段后(标志是测试缺陷报告库中所有的报告处于归档状态),由项目组和测试组长共同决定产品进入稳定期测试。稳定期测试版本之前的版本必须在显著位置标明为测试版字样。

(7) 稳定期测试期间所发现的缺陷报告也需要记录在测试缺陷报告库中,并在稳定期结束后由双方(有时可能也有市场方面的意见)共同决定对这些缺陷的处理方式。如果需要改动产品,则重新开始稳定期,否则通过稳定期测试。

(8) 测试组长对于通过稳定期测试的产品填写综合测试报告,测试中心依此发布产品发行通知。

(9) 测试小组对整个测试过程和产品质量进行总结和评价,形成文档并备案。同时,将测试过程中对测试设计的改动纳入基线。最后,组长整理并在指定地点保存相关测试用例、测试数据以及测试文档。

(10) 测试部门解散测试组。在解散测试组以前一定进行工作总结(每个人都要总结,总结测试过程中使用新的方法、技巧或者是某些方法、技巧的心得,以及测试过程中的失误),之后,将其形成文档或程序归档。

另外,在系统测试阶段,也将要求测试组进行一些常规内容测试(如早期的 Y2K 测试、病毒检查、裸机测试、加密检查、说明书检查等),并要求写入测试方案中。

8.3 测试过程管理工具

当前,测试过程管理工具主要分为商用和开源。在商用中,HP 的 Quality Center 是非常知名的,而国产知名工具要数北航的 QESuite;开源的则是以 TestLink 为代表了。

8.3.1 HP Quality Center 介绍

HP Quality Center 是基于 Web 的软件测试过程管理系统,用于各种 IT 环境和应用环境中的自动软件质量测试。它专门用于优化关键质量控制活动(包括需求、测试与缺陷管理、功能测试和业务流程测试),并使其实现自动化。在各个流程点上,管理人员可以看到质量控制活动的所在位置,从而在开发和测试应用的过程中,能够在管理和控制风险的同时,优化软件质量。

从 TD(TestDirector)测试管理套件,到 9.0 的 QC(Quality Center)质量控制中心,早期的 Mercury 团队敏锐地感觉到不能仅仅将“她”局限于测试领域,而应该将其上升到质量管理的领域,从名称的变更和部分实质性的变更开始,Mercury 为其确定了方向。

自 2006 年 HP 收购了 Mercury 之后,在 HP 集团整体逐渐重视软件业务的大背景下,

2010年年底QC 11.00正式命名为ALM(Application Lifecycle Management,应用程序生命周期管理)解决方案。从命名上可以直观地看出HP对"她"的定位,已经扩展到整个应用生命周期的全过程管理的解决方案。

ALM 11.00并非对QC的简单升级,"她"在此版本中包含了需求管理、开发管理、质量中心、性能中心等几大部分,显而易见HP已经将应用软件的整个生命周期管理和度量纳入其中。如HP Quality Center 11.0在以下5个方面能够很好地帮助用户达到重要目标:①通过对需求测试及缺陷检测的详细追踪改善应用质量;②根据风险级别调整和确定测试工作优先级,实现资源优化;③通过对应用交付流程的实时报告增强可预见性;④通过应用HP Sprinter,提高手动测试效率并增强创新性;⑤通过一个共享的需求、测试及缺陷检测的中央储存文件来增强可重复性。

1. QC的核心内容

QC是一个统一的、可扩展的平台,提供一致的、可重复的流程:①需求管理;②测试计划、调度和执行;③发布和周期管理;④缺陷管理;⑤流程和状态的实时报告;⑥同开发环境集成。

QC带来的好处有:①统一的平台,质量人员、开发人员和业务分析人员的分布式合作和交流;②上线前发现问题,从而降低成本;③管理发布流程,通过实时视图做出正式的上线决策。

2. QC的测试管理

QC是一个基于Web的测试管理工具,可以组织和管理应用程序测试流程的所有阶段,包括指定测试需求、计划测试、执行测试和跟踪缺陷。此外,通过QC还可以创建报告和图来监控测试流程。

QC是一个强大的测试管理工具,合理地使用QC可以提高测试的工作效率,节省时间,起到事半功倍的效果。

利用QC,我们可以:①制定可靠的部署决策;②管理整个质量流程并使其标准化;③降低应用程序部署风险;④提高应用程序质量和可用性;⑤通过手动和自动化功能测试管理应用程序变更影响;⑥确保战略采购方案中的质量;⑦存储重要应用程序质量项目数据;⑧针对功能和性能测试面向服务的基础架构服务;⑨确保支持所有环境,包括J2EE、.NET、Oracle和SAP。

使用QC进行测试管理包括4部分。

(1) 明确需求:对接收的需求进行分析,得出测试需求。

(2) 测试计划:根据测试需求创建测试计划,分析测试要点及设计测试用例。

(3) 执行测试:在你的测试运行平台上创建测试集或者调用测试计划中的测试用例执行。

(4) 跟踪缺陷:报告在你的应用程序中的缺陷并且记录下整个缺陷的修复过程。

3. QC的质量管理

惠普质量中心(HP Quality Center,以下简称QC)管理和控制质量流程,并在其应用环境中对软件测试实现自动化。它的功能可以满足管理发布流程和制定更有效的发布决策的需要。

QC可以提供一种稳定一致的、可重复的流程来收集需求、规划和安排测试进度、分析测试结果、管理缺陷和问题,从而使我们能更迅捷、更有效地部署高质量的应用。

QC是一款基于Web、适用于质量管理所有重要方面,包括需求、测试计划、测试用例、测

试执行和软件缺陷等的管理以及它们的独立应用。不仅可以把这些核心模块作为单独的解决方案来使用，也能将其集成到全球 Quality Center of Excellence（质量卓越中心）环境中来使用。

QC 能够为 IT 小组内部的高层次的交流和协作提供支持。无论正在协调多个 QA 小组的工作，或是正在一个大型的、分布式的卓越中心（Center of Excellence）中展开工作，QC 都可以帮助我们跨地域、跨机构地获取有益的信息。

使用 QC，产品研发机构中的各类团队就可以为统一的质量流程群策群力：

（1）业务分析人员定义应用需求和测试目标。

（2）测试经理和项目主管制定测试计划，并开发测试用例。

（3）测试自动化工程师创建自动化的脚本，并将脚本保存于存储器中。

（4）QA 测试人员运行手动测试和自动测试，汇报执行结果，并输入缺陷。

（5）开发人员登录数据库检查并修复缺陷。

（6）项目经理创建应用状态报告，并管理资源的分配情况。

（7）产品经理对应用发布的就绪状况做出决策。

8.3.2　北航软件所 QESuite

北京航空航天大学软件工程研究所研发的 QESuite 软件测试过程管理工具是基于 Web 面向软件产品的整个生命周期，实现对测试过程、测试对象、测试数据的有效管理，指导用户实施测试过程改进，满足开发企业对于测试管理的基本需求，是一个轻量级的测试过程管理工具，特别适用于对一个小型测试队伍（少于 30 人）的测试过程的管理，是中小型软件开发企业打造测试管理体系的有效工具。QESuite 系统具有以下功能特点。

（1）软件测试过程全面管理。可以同时管理多个正在进行的软件测试项目。同时可对软件产品从划分功能分类、分配相关人员、编写测试用例、执行测试用例到生成问题报告的整个测试流程进行系统、有效的管理。项目管理人员可以通过该系统随时了解、监控被测项目的执行进度和软件问题的处理状态，为测试人员和开发人员的工作提供有效的考核依据，保障软件开发和测试的顺利进行。

（2）被测软件产品功能分类机制确保测试覆盖率。能协助项目管理人员对被测软件产品的各功能区域进行缜密划分，从而可避免被测软件功能区域的重复或遗漏，同时可便于项目管理人员分配测试工作。

（3）对测试用例与软件问题报告的数据库管理。使软件开发企业可以脱离原有的纸张与电子表格等原始的文档记录方式，采用 Lotus Notes 或 SQL 等大型数据库管理方式。无论是开发人员、测试人员或项目管理人员都可以随时编写、修改和查阅测试用例和软件问题报告，并可对测试用例与软件问题报告进行长期保存，避免了测试用例与软件问题报告的流失。

（4）测试用例执行与软件问题处理过程的全程跟踪。可对测试用例的编写与执行情况进行全程记录，便于项目管理人员追踪测试用例在各个测试阶段的执行过程，及时调整测试策略与方法；并可记录软件问题从发现、分析到解决的整个状态转换过程和人员操作记录，便于项目管理人员追溯软件问题处理的各个过程，有助于进一步提高软件问题的处理质量与处理效率。

（5）统一的测试用例与软件问题模板。提供了统一的软件问题报告模板与测试用例模板，使测试人员能够更加准确、详细地编写测试用例与描述软件问题，保证了测试用例与软件

问题报告描述的一致性,便于对测试用例与软件问题的积累、分类与查询。

(6) 软件问题生命期状态的科学定义。科学划分的软件问题生命周期主状态及子状态,可以帮助用户详细记录、跟踪和管理软件问题的生命周期全过程。基于此软件问题生命周期状态转换图而定义的软件问题处理工作流,将测试部门与开发部门的工作结合在一起,将大大提高软件问题的处理效率与准确性。

(7) 实用的统计功能辅助管理决策。具有实用的统计功能,项目管理人员可以从各种角度建立分析统计报表,以便及时掌握测试用例的执行情况,软件问题的有效发现率、有效修复率和各项测试工作的进度,并进行全局管理。

(8) 强大的安全机制保障用户的数据安全。用户可以根据人员的分工和职能不同划分严格的权限,从而明确测试任务,并保证系统数据的安全。QESuite Notes 版更是构建在 Lotus Domino/Notes 的强大的安全机制基础之上,系统数据安全将更有保证。

(9) 支持不同的运行环境。QESuite 现有 Web 版和 Notes 版两种运行版本,可适应不同企业对运行成本和系统运行安全的不同要求。QESuite Notes 版基于 Lotus Domino/Notes 构建,充分利用了 Domino/Notes 的强大群组协同能力和强大的安全机制,适合于对系统数据安全性要求比较高的软件企业,也适用于已经拥有 Lotus Domino/Notes 平台的软件企业。而 QESuite Web 版基于 B/S 结构,运行环境简单,投入成本低。

8.3.3 TestLink(开源免费)

TestLink 用于进行测试过程中的管理,通过使用 TestLink 提供的功能,可以将测试过程从测试需求、测试设计到测试执行完整地管理起来,同时,它还提供了多种测试结果的统计和分析,使我们能够简单地开始测试工作和分析测试结果。而且,TestLink 可集成通用的 bug 跟踪系统,如 Bugzilla、mantis 和 Jira。

TestLink 是 sourceforge 的开放源代码项目之一。作为基于 Web 的测试管理系统,TestLink 的主要功能包括测试需求管理、测试用例管理、测试用例对测试需求的覆盖管理、测试计划的制定、测试用例的执行、大量测试数据的度量和统计功能等。

习题

1. 软件测试过程模型主要有哪些?它们之间有什么关系?各表明什么意思?
2. 如何在软件测试中运用软件测试过程模型?运用中要注意些什么问题?
3. 软件测试过程的概念,软件测试过程包含哪些活动和内容?怎样对软件测试过程进行度量?
4. 什么是软件测试过程成熟度?CMM 与 ALM 之间有什么关系?软件测试过程改进与软件过程改进存在什么样的关系?
5. 软件测试管理的流程,软件测试过程管理包括哪些基本内容?
6. 软件测试管理各阶段要完成的主要任务有哪些?需要编写哪些文档?
7. 怎样获取软件测试需求?在软件测试设计中,需要考虑哪些问题?
8. 怎样执行软件测试用例,分析软件测试结果,撰写软件测试文档?
9. 简述软件测试用例、测试数据与测试脚本三者的概念以及它们之间的关系。
10. 简述软件测试过程中的配置管理及组织管理思想、方法和技术手段。

第4部分

高级软件测试方法与技术篇

前面我们系统地介绍了传统的软件测试技术，这些技术是我们开展各种软件测试的基础。事实上，目前我们开展的软件测试无论是从形式上、内容上，还是从应用上都有非常大的变化，用到的软件测试方法与技术也发生着很大变化，如软件自动化测试、软件可靠性测试、软件安全性测试、软件本地化测试、面向对象软件测试等。这些软件测试方法与技术构成了我们所说的现代软件测试方法与技术。



第9章 软件测试自动化

长期以来，软件测试是整个软件生命周期中最薄弱的环节。测试软件也是一项艰苦的工作，它需要投入大量的时间和精力，因为在很多软件项目的测试中，测试人员的任务都是对被测软件进行手工测试。事实上，所有的测试活动都可以用传统的手工测试方式完成。

在传统的手工测试方法中，测试人员根据测试大纲中所描述的测试步骤和方法，由测试人员手工地输入测试数据，记录测试结果。手工测试的特点是能详细地执行软件的各个功能，测试速度由人控制，能够完整而从容地观察软件的运行情况并立即报告测试结果。

在手工测试中，有很大一部分测试其操作是重复性的、非智力创造性的，需要认真、细致、集中注意力的，且具有独立性的工作。计算机最适合代替人类去完成这些任务。因此，在这种状况下，大多数人会选择开发并使用工具，使工作更加轻松和高效。如单元测试中自动生成驱动程序、桩程序和部分测试用例，"白盒"测试的自动插桩及覆盖率统计，数据库应用系统测试的数据库记录自动生成，以及大量网络虚拟用户的生成等。测试工具虽然可以代替部分测试工作，使工作变得容易，工作质量得到提高，但不能使测试过程完全自动完成，仍旧需要人工干预。所以人们将这些软件测试工具集成起来，并做进一步开发，使得软件测试的启动、执行、结果分析等都不用人工干预——自己执行测试用例、查找软件的缺陷、分析收集测试信息、记录测试结果等，这就是软件测试中的自动化测试或称软件测试的自动化。

9.1 软件测试自动化概念

9.1.1 自动化测试的定义

所谓自动化测试就是执行由某种程序设计语言编制的自动测试程序控制被测软件的行为，模拟手工测试步骤，完成全自动或半自动测试。所谓全自动测试就是测试过程完全不需要人工干预，由程序自动完成测试的过程；所谓半自动测试就是在测试过程中需要人工输入数据或选择测试路径，再由自动测试程序完成测试的过程。

虽然软件测试的工作量很大，但它却是在整个软件开发过程中最有可能应用计算机进行自动化的工作，原因是测试的许多操作是重复性的、非智力创造性的、需要细致集中注意力的工作。计算机最适合于代替人类去完成这些任务。

因此对于那些步骤与方法相对固定的测试可以采用自动化测试方法。自动化测试的优点是可以大规模地提高测试效率，减少测试工作量，具有可重复性，可以精确地再现以前的测试步骤，有利于进行回归测试，可以降低人为的操作失误和对测试人员的技术要求，从而降低测试成本，大大节约软件产品整个开发周期的费用，提高软件的质量。

自动化测试通常比手工测试经济得多，其开销只是手工测试的一小部分。第一次自动执

行相同的测试用例时，由于在自动化上花费了较多的工夫，其经济性较低。但是当自动化测试运行多次后，就比手工执行相同的测试要经济得多。

自动化测试相比手工测试其修改性比较低，这是因为在自动化测试中，必须要增加额外的维护开销，以对修改后的测试用例重新进行自动化。

9.1.2 适合于自动化测试的相关活动

软件测试过程中一般包括测试策划、测试设计、测试执行和测试总结等活动。

(1) 测试策划的主要任务是：①进行测试需求分析，即确定需要测试的内容或质量特性；②确定测试的充分性要求；③提出测试的基本方法，确定测试资源和技术要求；④进行风险分析与评估，制定测试计划(包括资源计划和进度计划)。

(2) 测试设计的主要任务是：①依据测试需求，分析并选用已有的测试用例或设计新的测试用例。②获取并验证测试数据。③根据测试资源、风险等约束条件，确定测试用例执行顺序。④获取测试资源，开发测试软件。⑤建立并校准测试环境。⑥进行测试就绪评审，主要评审测试计划的合理性和测试用例的正确性、有效性和覆盖充分性，评审测试组织、环境和设备工具是否齐备并符合要求。在进入下一阶段工作之前，应通过测试评审。

(3) 测试执行就是执行测试用例，获取测试结果，分析并判定测试结果。同时，根据不同的判定结果采取相应的措施。对测试过程的正常或异常终止情况进行核对，并根据核对结果，对未达到测试终止条件的测试用例，决定是停止测试还是需要修改或补充测试用例集，并进一步测试。

(4) 测试总结主要是整理和分析测试数据，评价测试效果和被测软件项，描述测试状态。如实际测试与测试计划和测试说明的差异、测试充分性分析、未解决的测试事件等；描述被测软件项状态，如被测软件与需求的差异、发现的软件差错等；最后，完成软件测试报告，并通过测试评审。

从上述对这 4 个活动的分析中，我们很显然可以确定有些测试活动不适宜进行自动化，有些则可以进行。如测试策划活动主要是测试需求分析，测试计划活动则是标识测试条件和设计测试用例，测试总结活动是对最终测试结果的一个分析和总结性工作，这些均属于智力性的管理活动或开发活动，就整体而言只执行一次，是不适宜进行自动化的。而测试执行活动主要是执行测试用例和检查测试结果，主要是机械活动，一般要执行多次，比较适合进行自动化。

此外，如果一个应用软件想增加一些新的功能，或者测试中发现软件有错误，修复 bug，经常会推出产品新的版本。在推出的过程中，对修改后的软件，为确保不引起其他错误，我们知道，除了测试修改过的模块外，每次还都要重复测试有关联的模块，这样很多时候会做大量的重复工作，这也就是我们所说的回归测试。回归测试将重复测试执行和结果比较与检查，这使测试人员很疲惫而且也达不到测试效果。对这些重复的活动进行自动化是合适的。

自动化功能测试工具可以创建整个测试生命周期的可重用模块，同时还能覆盖大部分的系统测试，更主要的是录制好脚本以后，自动去执行，机器去操作，减少了人为主观的错误，同时使测试人员解脱出来，专注新的模块。自动化测试最大的价值在于回归测试。在产品修改或升级提交过来之后要执行回归测试，自动化测试工具能够节省人力、时间和金钱。

9.1.3 自动化测试的优点

自动化测试与手工测试相比具有手工测试不能比拟的优点。

(1) 执行一些手工测试不可能或很难完成的测试。例如，对于 200 个用户的联机系统，用

户手工进行并发操作是几乎不可能的,但自动化测试可以模拟来自 200 个用户的输入。

(2) 提高测试的效率。在需要多次执行的情况下,自动化测试不需要测试人员每次都重复相同的过程。自动化测试建立起来后,就可以多次重复执行,大大提高了测试的效率。测试人员从繁重的测试执行中解脱出来,将更多的精力用来设计更多、更好的测试用例。

(3) 提高测试的准确性,降低对测试人员的技术要求。手工测试需要测试人员理解测试步骤和被测试软件,并按照测试步骤一步一步地执行。在执行过程中,测试人员难免会犯这样或那样的错误,这些都会影响测试的准确性。而自动化测试建立起来之后,测试人员只需要执行自动测试用例并在必要时对输出结果进行一定的检查即可。这大大提高了测试准确性,同时也降低了对测试人员的技术要求。

(4) 可实现无人照料测试。自动化测试还可以实现无人照料测试,充分利用休息时间进行测试。这样可以更加合理地利用测试资源,进行更多的测试。

(5) 具有一致性和可重复性。可以利用自动化测试重复多次相同的测试,这样就可以保证测试的一致性。而这在手工测试中很难进行。

(6) 有利于进行回归测试。回归测试往往需要重复以前进行过的测试,自动化测试具有良好的可重复性,使得回归测试比较容易进行。

(7) 缩短测试的时间。一旦实现了测试自动化,就可以比手工测试更快地执行测试,缩短测试的时间,可以更快地将软件推向市场。

9.1.4 自动化测试的局限性

软件测试的自动化能大大降低手工测试工作,但是它不能完全替代手工测试。达到 100%的测试自动化只是一种理想的目标,是很难实现的。因为不仅代价相当昂贵,而且操作上也是几乎不可能实现的。

一般来说,一个软件测试自动化达到 40%～60%的程度已经是非常好的了,达到这个级别以上将增加与测试用例管理相关的维护成本。

在软件测试中,如果没有很好的测试基础,特别是自动化测试的基础,那么为了推行自动化测试,其前期所要做的工作是相当庞大、相当复杂的。如自动化测试脚本的编写,自动化测试过程的确定和实施,以及为实施测试自动化所必须进行的多方面培训(包括测试流程、缺陷管理、人员安排、测试工具使用等)等都是必须开展的工作。否则引入自动化测试只会给软件组织或者项目团队带来更大的混乱。

对于周期短、时间紧迫的项目不宜采用自动化测试。如对一个项目周期很紧的新项目进行功能测试,如果临时分配几个人,按测试方案进行手工测试的效率可能要比用自动化测试工具采取录制脚本/编制脚本进行测试的效率好得多。否则会因为可能需要测试框架的准备和实施、大量的测试用例或测试脚本的编写而被拖垮。

另外,软件测试自动化能提高测试效率,快速定位出测试软件的功能和性能的缺陷,但是它不会如同人脑那样能创造性地发现脚本设计里的缺陷,如果测试设计者在测试设计时出现了缺陷,测试工具是不会发现的。

从以上分析可以看出,自动化测试不是万能的,它也存在一定的局限性。软件测试自动化不能解决所有的问题,例如:

(1) 软件自动化测试并不能代替人的工作,我们不要期望将所有的测试活动或测试进行自动化。软件测试工具不能发现所有的问题,测试人员还需要做大量的工作。

(2) 软件测试自动化可能降低测试的效率。当测试人员需要进行很少量的测试,而且这

种测试在以后的重用性很低时，花大量的精力和时间去进行自动化的准备往往是得不偿失。因为自动化的收益一般要在多次重复使用中才能体现出来。

(3) 缺乏测试经验。如果测试的组织差、文档较少或不一致，则自动化测试效果就会比较差。

(4) 技术问题。毫无疑问，商用软件自动化测试工具是软件产品。作为第三方的技术产品，如果不具备解决问题的能力和技术支持或者产品适应环境变化的能力不强，将使得软件自动化测试工具的作用大大降低。

因此，我们对软件自动化测试应该有正确的认识，它并不能完全代替手工测试。不要期望有了自动化测试就能提高测试的质量，如果测试人员缺少测试的技能，那么测试也可能会失败。

对于那些步骤与方法相对固定的测试，可以采用自动化测试方法。自动化测试的优点是可以大规模地提高测试效率，减少测试工作量，具有可重复性。可以精确地再现以前的测试步骤，有利于回归测试，可以降低人为的操作失误和对测试人员的技术要求，从而降低测试成本，大大节约软件产品整个开发周期的费用，提高软件的质量。

9.2 软件测试自动化框架

自动化测试在过去的20年中已经有了很大的发展。最初的测试工具只提供了简单的捕捉/回放功能：记录并回放键盘按键和点击鼠标等操作，然后捕捉和比较屏幕。这些测试方法虽然最容易应用，但是几乎不可能维护。捕捉/回放工具最终被功能和灵活性更强的测试脚本工具代替。

但是，脚本工具也有自己的问题。它们实现起来需要很强的开发技术和经验，同时，不确定它们是一定可以维护的。更糟糕的是高度个性化的脚本工具技术，加上没有什么文档记录，最后的结果经常是重写包含成千上万行代码的脚本库，成本开销巨大。

后来，一种新的自动化测试产品出现了。它可以减少实现和维护的成本，使测试人员可以把精力集中在应用程序的测试用例设计上，而不是开发我们的测试。这些工具提供预先写好的测试框架，可以极大地减少，甚至消除学习和使用脚本语言的需要。这个测试产品就是自动化测试框架。

9.2.1 自动化测试框架概念

在了解什么是自动化测试框架之前，先了解一下什么叫框架？框架是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法；另一种定义认为，框架是可被应用开发者定制的应用骨架。前者是从应用方面而后者是从目的方面给出的定义。从框架的定义可以了解，框架可以是被重用的基础平台，框架也可以是组织架构类的东西。

如果将框架的概念应用于测试领域，就是我们常说的测试框架。搭建测试框架的最终目的是花少量的资源来完成尽可能多的测试任务，所以测试框架的建立以及框架的重用性方面是测试人员最为关注的。如果期望测试能够自动化开展或进行，则要搭建自动化的测试框架。

在很多文章中介绍自动化测试框架概念时，将自动化测试框架看做由一些假设、概念和为自动化测试提供支持的实践组成的集合，是指测试的驱动、执行和管理策略。

1. 什么是测试框架

测试框架是一组自动化测试的规范、测试脚本的基础代码，以及测试思想、惯例的集合。建立测试框架的优点在于：①减少冗余代码，提高代码生产率，提高代码重用性和可维护性，

提高开发速度,提升测试代码的执行效率;②提高软件代码质量,同时引入重构概念,让代码更干净和富有弹性;③提升系统的可信赖度,作为回归测试的一种实现方法支持修复后"再测试",确保代码的正确性。

2. 测试 Harness

在讨论软件测试自动化框架时,经常会见到 Test Harness 一词。所谓 Test Harness 就是在软件测试中,Test Harness 或自动化测试框架是一个包含了软件工具和测试数据(这些数据已经得到配置)的集合,用以对被测程序进行测试,使之在不同的条件下运行,并监控它的行为状态和输出结果。Test Harness 有两个主要的组成部分:测试执行引擎和测试脚本储存库(Repository)。因此,Test Harness 能够帮助我们实现自动化的测试。

当我们编写测试用例时,无论是单元测试用例以及测试驱动式开发中的测试用例,还是回归测试用例或是其他类型的测试用例,我们都会重复地做一些事情。这些事情包括构建应用基础,测试实施以及测试结果报告方式。每次要测试程序的一个新特征时,都要重复地写这些东西,或者一次把它们写好,然后每次都修改它们。因此,Test Harness 是帮助我们处理重复编写测试用例时所面临各种问题的系统平台。

基本的 Test Harness 至少要包括下面几个要素:应用基础、测试实施和结果报告。同时也可以包含图形用户界面、日志系统与测试用例脚本。我们必须注意:Test Harness 通常只为某个语言或运行时而开发的,如 C/C++、Java、.NET,很难去构建支持各种语言或运行时的一个通用的 Test Harness。

Test Harness 需要做的第一件事就是配置好应用程序支持其测试。不同的操作系统其应用开发是不一样的。例如,在 Windows 平台上,如果你要用到 GUI,你需要类似 message pump 和 message loop 的消息处理机制。该消息处理机制专门处理应用程序与操作系统的交互。Test Harness 要完成好应用平台的配置,使之能够启动应用程序的代码,打开所需文件,以及选择测试用例的运行。

Test Harness 需要做的第二件事就是确定测试实施的方式。大多数情况下,测试用例仅是一个功能或是方法。这个功能要完成测试用例所有要做的工作。对于 API 调用测试,检查测试结果,并告知测试是否通过。Test Harness 采用一种规范的途径向系统提交测试用例并提供标准的接口来启动测试用例。以最简单的系统举例,C/C++的 Harness 通过向表格添加函数指针的方法向系统提交测试用例。Test Harness 还要提供一些额外的服务,以允许系统以特定的次序调用测试用例,或通过独立的线程来调用测试用例。

Test Harness 需要做的第三件事就是生成测试结果报告。即为测试用例提供输出信息和报告测试是否通过的途径。在大多数的 Test Harness 中,测试用例向窗口终端输出相关信息。如果系统做得更好,则自动地显示每个测试用例的名称和测试的结果。通常要有一个测试用例多少通过和多少没有通过的统计,这可以正文的方式或直方图的形式给出。好一点的系统则内部提供有日志功能,该功能以规范的方式为测试用例输出跟踪信息,向用户提供每次测试用例执行的详细情况。Test Harness 也可以将简单的日志以 XML 格式输出到一个正文文件中,或以记录的形式存储到数据库中。

在微软公司里,很多小组开发自己的 Test Harness,以满足特定项目测试的需要。例如,一个叫 Shell98 的 Test Harness,较特殊的是,它包含有各种设施,均是一些可以免费获得的 Test Harness,如 cppUnit、nUnit 以及 jUnit 的 xUnit 系列。这些软件用于单元测试,功能有限。其中 cppUnit 非常基础,简单好用,而 nUnit 灵巧朴实。xUnit 这类 Test Harness 没有日志功能,也不能控制测试用例执行的先后顺序。

3. 自动化测试框架

有了 Test Harness 概念后，我们很容易讨论什么是软件测试的自动化框架了。

1）自动化测试框架定义

所谓自动化测试框架，即是应用于自动化测试所用的框架。按照框架的定义，自动化测试框架要么是提供可重用的基础自动化测试模块，如 selenium、watir、QTP 等，它们主要提供最基础的自动化测试功能，比如打开一个程序，模拟鼠标和键盘来单击或操作被测试对象，最后验证被测对象的属性以判断程序的正确性；要么是可以提供自动化测试执行和管理功能的架构模块，如 Robot、STAF、QC 等，它们本身不提供基础的自动化测试支持，只是用于组织、管理和执行那些独立的自动化测试用例，测试完成后统计测试结果，通常这类框架一般都会集成一个基础自动化测试模块，如 robot 框架就可以集成 selenium 框架。

所以自动化测试框架的定义为：由一个或多个自动化测试基础模块、自动化测试管理模块、自动化测试统计模块等组成的工具集合。

2）自动化测试框架结构

自动化测试框架一般分为上下两层，上层是管理整个自动化测试的开发、执行以及维护，在比较庞大的项目中，它体现重要的作用，它可以管理整个自动测试，包括自动化测试用例执行的次序、测试脚本的维护，以及集中管理测试用例、测试报告和测试任务等。下层主要是测试脚本的开发，充分地使用相关的测试工具，构建测试驱动，并完成测试业务逻辑。

为了能开展自动化测试的工作，首先需要基础设施（Infrastructure）来支撑测试工具的运行，这包括 Web 服务器、邮件服务器、FTP 服务器等。其次是执行自动化测试，要有一套机制来保证测试脚本的执行。具体地说，就是先建立测试环境，创建和执行测试套件，然后获取执行状态并给出测试结果报告。这些都是下层必须支持的。

9.2.2 常用的自动化测试框架

在软件测试自动化实施中，面临自动化测试框架的选择或开发。那么采用什么样的原则来选择或开发所需要的自动化测试框架呢？

当然，首先要重视测试自动化，在确定实施测试自动化后，一定要锲而不舍地当做主要工作来做；其次，测试设计的开展和测试自动化框架的构建是分别独立的，而且测试框架最好不要依赖被测程序；最后，测试框架应该易学、易用，易于扩展、升级和维护。

1. 自动化测试框架的分类

自动化测试框架分类方法有多种。按框架的定义来分，自动化测试框架可以分为基础功能测试框架、管理执行框架；按不同的测试类型来分，可以分为功能自动化测试框架、性能自动化测试框架；按测试阶段来分，可以分为单元自动化测试框架、接口自动化测试框架、系统自动化测试框架；按组成结构来分，可以分为单一自动化测试框架、综合自动化测试框架；按部署方式来分，可以分为单机自动化测试框架、分布式自动化测试框架。

2. 常用自动化测试框架类型介绍

这里介绍 5 种比较常用的自动化测试框架类型。可以根据实际需要考虑采用其中的一种测试框架类型。同时，我们也可通过这些框架的学习，了解自动化测试框架使用方法。

1）测试脚本模块化框架

测试脚本模块化框架需要创建能够代表被测程序模块、片段（Section）和函数对应一个个小而独立的脚本。然后用一种分级的方式将这些小脚本组成更大的测试，实现一个特定的测试用例。该框架与后面介绍的几种自动化测试框架相比，是最容易掌握和使用的。它很好地

支持着面向对象程序设计或结构化程序设计中的模块化、抽象、封装及信息隐藏等编程思想及原则在软件开发中的应用。测试脚本模块化框架由于应用了抽象或封装的原则，从而大大提高了支持自动化测试的测试集(Test Suite)的可维护性和可测量性。

2) 测试库框架

测试库框架与模块化测试框架很类似，并且具有同样的优点。不同的是测试库框架把被测程序分解为过程和函数而不是脚本。这个框架需要创建描述模块、片断以及被测程序的功能库文件(如 SQABasic Libraries、APIs、DLLs 等)。

3) 数据驱动测试框架

数据驱动(Data Driven) 测试是一个框架。在这里测试的输入和输出数据是从数据文件中读取(数据池、ODBC 源、CVS 文件、Excel 文件、DAO 对象、ADO 对象等)，并且通过捕获工具生成或者手工生成的代码脚本给加载到变量。在这个框架中，变量不仅被用来存放输入值，还被用来存放输出的验证值。通常在整个程序中，测试脚本用来读取数值文件，记载测试状态和信息。这类似于表驱动测试。在表驱动测试中，它的测试用例是包含在数据文件而不是在脚本中，对于数据而言，脚本仅仅是一个“驱动器”，或者是一个传送机构。然而，数据驱动测试不同于表驱动测试，尽管导航数据并不包含在表结构中。在数据驱动测试中，数据文件中只包含测试数据。这个框架意图减少你需要执行所有测试用例所需要的总的测试脚本数。数据驱动需要很少的代码来产生大量的测试用例，这与表驱动极其类似。

4) 关键字驱动或者表驱动的测试框架

关键字驱动或者表驱动的测试框架是一种独立于应用程序的自动化框架，在处理自动化测试的同时也要适合手工测试。关键字驱动的自动化测试框架建立在数据驱动手段之上，表中包含指令(关键词)，而不只是数据。这些测试被开发成使用关键字的数据表，它们独立于执行测试的自动化工具。关键字驱动的自动化测试是对数据驱动的自动化测试的有效改进和补充。关键字驱动的自动化测试的整个过程所包含的功能都是由关键字驱动的，关键字控制了整个测试过程。

5) 混合测试自动化框架

综合以上两种自动化测试框架，取长补短，弥补各自的不足：以数据驱动的脚本作为输入，通过关键字驱动框架的处理得到测试结果，完成自动化测试过程。这样可以使数据驱动的脚本利用关键字驱动框架通常所提供的库和工具。这些框架工具可以使数据驱动的脚本更为紧凑，而且也不容易失败。

9.2.3 基于 API 测试的分布式测试框架

随着信息技术的迅猛发展，系统平台日趋复杂多样，出现了实时系统、嵌入式系统、分布式系统、数据库系统以及各种应用支持技术(如面向服务的技术、云计算技术、移动技术等)。这些系统和技术虽然都有各自不同的技术特点，但它们均提供了支持应用开发的 API。为确保这些 API 的正确、可靠，必须对这些 API 进行全面的测试。随着软件技术的发展，被测 API 的种类日益多样、形势日趋复杂，为之编写测试用例、设计测试数据的要求也不断增加和膨胀，人们很难对它们进行管理和执行驱动，这表明人们对测试驱动器的要求也越来越高。同时运行于多机系统的可满足远程及分布式测试需要的测试驱动器已是必然要求。因此，除了上面所讲的传统上的几种常用的测试框架外，一种新的测试框架也随之产生：基于 API 测试的分布式测试框架。

1. 分布式测试总框架

图 9-1 是所讨论的分布式测试框架的总体结构框架图，它简单描述了在执行分布式测试

用例时各功能部件之间是如何关联的。部件间的箭头表示通信和交互关系，部件与文件之间的箭头表示输入输出关系。

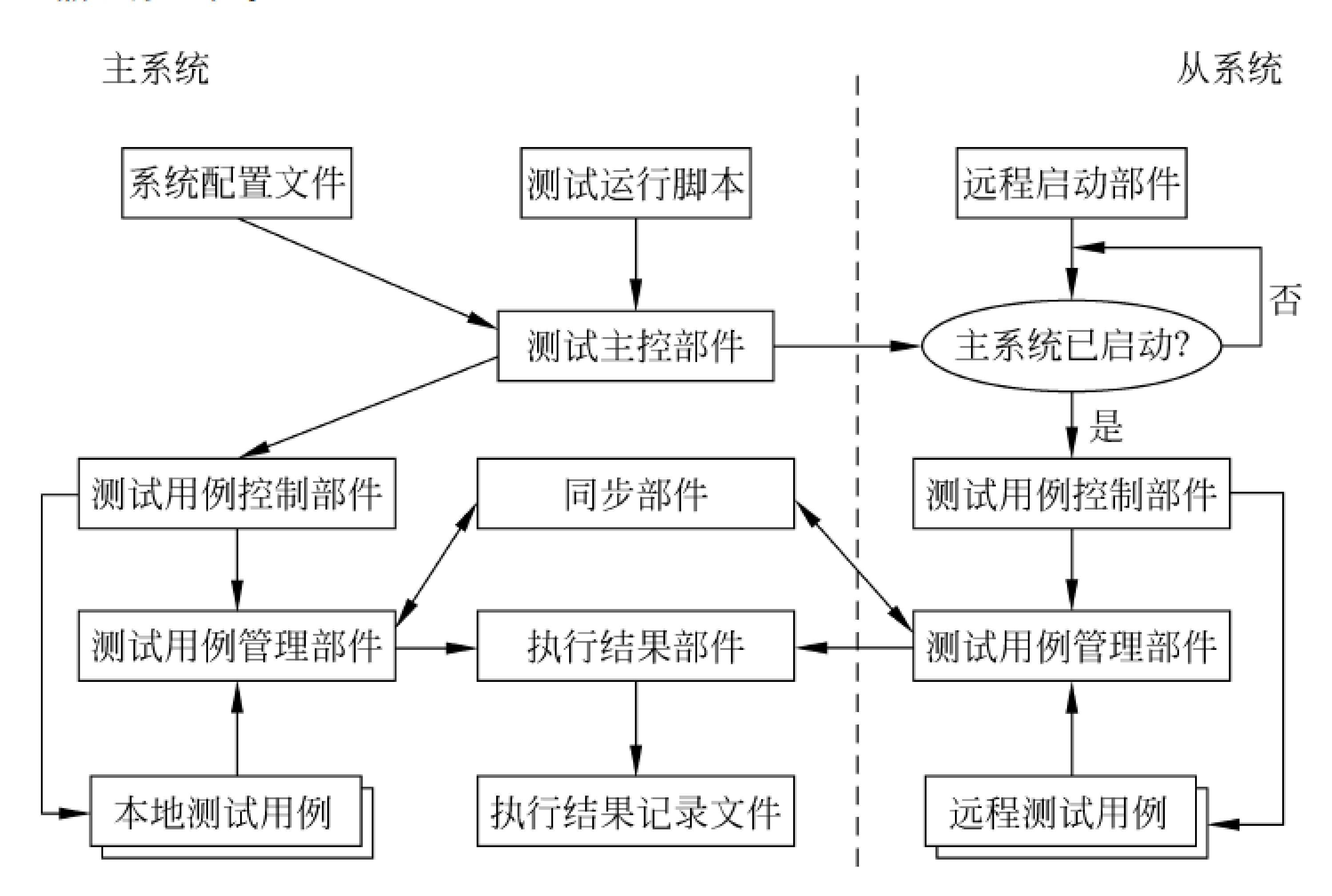


图 9-1 分布式测试总体结构框架图

1）分布式测试框架的工作原理

在图 9-1 中，主系统和从系统合称参与系统。主系统是参与系统中运行测试主控部件的系统，负责管理所有参与系统的测试工作，参与系统中有且只有一个主系统；从系统是参与系统中运行远程启动部件的系统，只负责执行本机上的测试任务，参与系统中可以有零到多台从系统。系统配置文件定义了每个参与系统的环境参数和系统标识符。测试运行脚本记载了所有的测试任务，描述了所有的测试用例的测试流程，是测试的执行依据。测试用例由用户提供的测试代码和测试用例管理部件提供的工具 API 按照一定的语法格式链接构成。结果文件记录了测试执行过程中产生的测试用例输出信息和测试执行信息。在完成所有的测试任务后，用户将结果文件上记录的测试结果与原先预想的执行结果相对比，从而得出正确的测试结论。测试主控部件的功能是读取系统配置文件和测试运行脚本，控制参与系统上的测试用例控制部件依次执行脚本上的测试任务。远程启动部件的功能是等待主系统启动，将对本机上的测试用例控制部件的控制权转交给主系统上的测试主控部件。

测试用例控制部件的功能是接收测试主控部件传送的测试任务，驱动测试用例管理部件依次编译执行测试任务中的测试用例。测试用例管理部件并不是一个功能实体，它由一组供测试用例使用的工具 API 和被这些工具 API 及测试用例控制部件调用的功能函数构成。工具 API 与用户提供的测试代码相结合构成测试用例。同步部件协助完成各测试用例管理部件之间进行的同步操作。执行结果部件负责接收由所有参与系统的测试用例管理部件发送出的输出信息，对信息进行适当的过滤和处理后，记录到结果文件中。

2）分布式测试框架的特点

此分布式测试框架在结构上主要有以下两大特点：

（1）采用客户机/服务器结构。主系统扮演服务器的角色，从系统扮演客户机的角色。服务器是此分布式测试框架中不可缺少的成分，客户机的数量根据测试需要可以是零到多台。服务器不仅可以参与执行本机上的测试用例，还可以通过测试主控部件向所有参与系统的测试用例控制部件分配测试任务，管理所有参与系统的测试工作。客户机只负责执行本机的测试任务。测试用例管理部件通过服务器上的同步部件与其他参与系统上的测试用例管理部件

进行同步。参与系统上的测试用例的执行结果全部发送到服务器上的执行结果部件，形成统一完整的测试结果。采用客户机/服务器结构是支持分布式测试和远程测试的基础。

（2）采用对称的框架结构。主系统与从系统上的测试用例控制部件和测试用例管理部件是完全一样的。由于在测试过程中只有这两个部件直接参与处理测试用例，因此主从系统上的测试用例的处理逻辑都是一样的。其优点是：在设计测试用例时不用考虑主从系统上的执行差异；在主系统或从系统上所执行的是非分布式测试用例还是分布式测试用例这个问题变得不再重要。采用对称的框架结构提高了测试框架的应用价值。

3）分布式测试框架的应用

此分布式测试框架可以同时应用于分布式测试、非分布式测试和远程测试。

（1）非分布式测试：执行非分布式测试用例，参与系统中只有主系统，不存在从系统。此时，主系统上的同步部件不工作，测试主控部件和执行结果部件只与主系统上的测试用例管理部件交互。

（2）远程测试：执行非分布式测试用例，参与系统中包含主系统和至少一个从系统。主系统可以不运行任何测试用例，每个从系统都必须运行一组测试用例。当主系统不参与测试用例的执行时，主系统的测试用例控制部件和测试用例管理部件不工作。在进行远程测试过程中，主系统上的同步部件不工作，参与系统并发执行本机上的测试用例，并发执行的测试用例之间不存在同步或交互操作。

（3）分布式测试：执行分布式测试用例，参与系统包含主系统和至少一个从系统。主系统可以不运行任何测试用例，每个从系统都必须运行一组测试用例。当主系统参与测试用例的执行时，其执行框架与图 9-1 所示完全相同；当主系统不参与测试用例的执行时，主系统的测试用例控制部件和测试用例管理部件不工作。由于采用了对称的框架结构，运行于主从系统上的分布式测试用例不需要区别对待。

2. 测试结构

测试结构包括三部分的内容，即测试用例的结构、测试运行脚本和测试用例的执行。

1）测试用例的结构

测试用例由用户提供的测试代码和供测试代码使用的工具 API 组成。工具 API 由测试用例管理部件提供，在功能上主要有三类：①通过同步部件进行同步操作；②通过执行结果部件向结果文件写测试信息；③配合测试用例管理部件的执行逻辑调用测试用例管理部件中的功能函数，驱动测试用例管理部件的执行。

测试用例以文件的形式存在，可以被编译成可执行程序。测试运行脚本中有多个测试任务，一个测试任务由一组测试用例组成，每个测试用例中包含多组测试代码，每组测试代码与穿插在其中的工具 API 构成测试目的。测试任务由测试主控部件管理；测试用例由测试用例控制部件管理；测试目的由测试用例管理部件管理，测试目的通过调用其中的工具 API 驱动测试用例管理部件的部分执行功能。

2）测试运行脚本

测试运行脚本是测试框架中的一个重要部分，是测试的执行依据。测试主控部件从测试运行脚本中获知要测试哪些测试用例以及如何开展测试。脚本技术的引入使得分布式测试框架具备了自动化测试能力。复杂的测试脚本结构不仅可以实现测试自动执行，还可以实现测试结果自动比较。测试人员可以根据测试需要设计测试运行脚本的语法和结构。如图 9-1 所示的测试框架中，任务名、执行命令、系统标识、执行元素是脚本应包含的最小内容。其中任务名用于标识一个测试任务；执行命令指示出在命令范围内的测试用例要进行的测试行

为，如分布式测试、并行测试、远程测试等；系统标识是参与系统的网络标识名；执行元素是测试任务中的测试用例的路径。

3）测试用例的执行

图 9-2 简单描述了本机上测试用例的执行流程。测试主控部件从测试运行脚本中逐个获取测试任务，并将测试任务中的测试用例的路径传递给测试用例控制部件。测试用例控制部件先调用测试用例管理部件编译这些测试用例，然后逐个调用这些测试用例，驱动它们的执行。如果是分布式测试用例，在测试用例中的部分测试目的之前会事先设定一个同步点，在同步点处通过调用用于进行同步操作的工具 API 来驱动测试用例管理部件通过同步部件与其他参与系统上的测试用例管理部件进行同步操作；在所有的参与系统都执行到此同步点后，开始执行该同步点之后的测试目的。在执行测试目的过程中，测试目的会调用它们中的工具 API，进行分布式数据发送、向结果文件写信息等操作。如果在执行分布式测试用例时，某个参与系统的测试目的发生异常，则结束此测试用例的执行，各参与系统准备下一个测试用例的执行。在结束所有的测试用例的执行之后，测试结束。

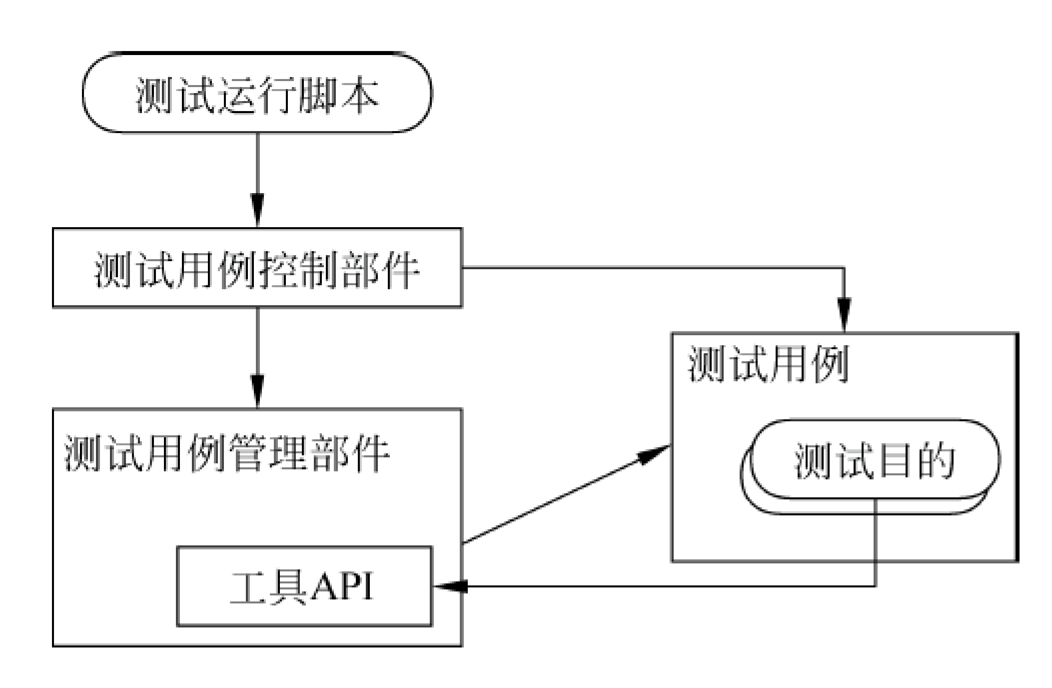


图 9-2 测试用例执行流程简图

3. 测试同步

下面结合所讨论的分布式测试框架介绍一种同步机制，具体的实现测试人员可以根据自己的需要在此基础上进行设计。

这里所讨论的分布式测试框架所进行的同步主要有两种情况：①在各参与系统上的测试用例管理部件之间通过交互部件对测试目的进行的同步操作，这种同步是在测试进行过程中固定同步点自动进行的，是可以预料到的，称为自动请求同步；②在进行分布式测试时，各参与系统上的分布式测试目的在试图发送和接收数据时所进行的同步，这种同步是根据测试需要，由用户在测试用例中随机设置的，无法事先预料，称为用户请求同步。

每一个自动请求同步伴随有同步点编号、系统 ID 号（即系统标识）、同步表决和超时值这几个请求参数；用户请求同步比自动请求同步应多一个同步请求数据和一个接收/发送标记。在设计测试用例时，每一个同步点都要赋予一个同步点编号，相同同步点编号的同步请求才能彼此进行表决；系统 ID 表明想要进行同步的参与系统号，在设计它时，可以采用一个列表结构；同步表决可以是 yes 或 no，表明发送请求的系统是否准备就绪；超时值限制了发送请求的系统等待应答的最长时限，如果超过此时限还没有得到最终结果，系统就认为此次请求失败；同步请求数据用于记录要发送或接收的数据；接收/发送标记表明系统是想接收数据还是想发送数据。

想进行同步的系统向同步部件发送同步请求。同步部件等到所有的系统都提交了请求后通知所有参与系统同步的最终结果。如果所有进行同步的系统的同步表决都为 yes 且没有系统因超时而退出同步，则同步最终结果为成功，否则为失败。如果是用户请求同步且同步成功，同步部件将把接收的数据发送到正确的参与系统上。

4. 松散集成

优秀的测试框架应具备良好的兼容性，具有管理第三方测试工具的测试用例及其测试结果的能力。针对这一要求，结合前面所讨论的分布式测试框架的特点，给出两个简单的解决方法。

(1) 第三方测试用例的驱动通过在测试运行脚本中加入该测试任务实现,测试用例的执行由测试用例控制部件调用第三方测试工具执行。在每次调用第三方测试工具执行测试用例时,由于第三方测试用例不会调用应用本测试框架开发的工具 API,测试用例管理部件不工作。第三方测试用例的输出不经过执行结果部件,结果文件只能记录第三方测试工具的结束状态。第三方测试用例的执行结果信息只能在第三方测试工具的结果文件中阅读。

(2) 在上述策略的基础上开发一个结果文件转换部件,该部件的职能是在第三方测试工具将测试结果输出到第三方结果文件时,产生一个临时的结果文件,将记录到第三方结果文件中的测试结果按照自己的风格翻译到临时结果文件中,用户不必再学习第三方结果文件的语法。

5. 测试环境

由于采用了对称的框架结构,且测试部件、同步部件和结果执行部件只在主系统上有,因此,此处讨论的分布式测试框架可以独立于平台和环境,即支持跨平台的分布式测试。另外,我们应注意,主从系统的确定与测试环境无关,只在于测试开始时本机上运行的是测试主控部件还是远程启动部件,启动不同的部件本机就会履行不同的测试职能。

9.3 自动化测试技术

自动化测试技术按其机制可以分为侵入式和非侵入式,侵入式测试技术采取某种方式修改内部代码或者控制其运行环境;而非侵入式测试技术用于监视和检查软件,而不修改软件内部结构或者代码。

自动化测试主要采用自动化工具提供的测试脚本完成针对目标应用程序的测试。测试脚本是某种采用特定语言编写,并在特定系统环境下实现的代码。根据测试功能的复杂程度,测试脚本可以是需要借助其他语言进行解析的文本,可以是简单的"批处理"命令,也可以是相对复杂的类似于 TCL 语言的功能强大的脚本语言程序片段。当前普遍使用的测试脚本主要由三种方式产生。

(1) 人工编辑测试脚本。与普通的编程原理类似,人工编辑测试脚本就是采用某种特定的编程语言,编写一系列能够在特定环境和平台下运行的代码(如批处理脚本),然后运行这些代码,达到自动测试的目的。

(2) 测试工具通过对被测程序进行逆向工程自动产生测试脚本。这种方式主要用于单元测试自动化技术中。对于采用面向对象方法设计的目标应用程序,单元就是程序中的各个类,针对这些类的单元测试采用这种技术生成测试脚本,首先要提供被测单元的源程序代码,在该代码单元内人工选择需要测试的方法,生成一个或者多个测试脚本。

(3) "录制/回放"脚本技术。这种方式属于大多数 GUI 自动化测试工具都使用的主流测试自动化技术,主要是由测试人员在测试工具平台环境下,手工对目标应用程序进行一次用例测试,由测试用例记录下手工操作的步骤和对象,作为测试脚本运用到以后的多次机械重复测试中去,以提高测试效率,减少重复测试工作量。

目前,软件测试自动化主要集中在软件测试流程的管理自动化和动态测试的自动化,如功能测试自动化和性能测试自动化方面,还有是少部分的静态测试,如代码查验,它们常常比较容易从开发过程中剥离出来。

9.3.1 脚本技术

测试脚本是交互应用或部分非交互应用的自动化测试中必要的组成部分。脚本是一组测

试工具执行的指令集合，也是计算机程序的一种形式。大多数的测试执行工具提供的脚本语言是非常有效的编程语言，但是即便如此，当脚本中的信息越来越多时，脚本出错的概率也就越来越大。因此我们需要使用不同的脚本技术来减少脚本的大小、数量和复杂度。

如同软件一样，脚本的好坏依赖于个人的编程技术，也依赖于实现对象。快速粗略的方法是尽可能地录制，或是复制其他脚本技术拼凑在一起，更好的一些方法是需要对拙劣的脚本进行设计和编程。由于脚本是测试体系中的一个关键部分，因此保证脚本的质量也就是保证软件测试的质量。图 9-3 给出了脚本开发成本与维护成本之间的关系。

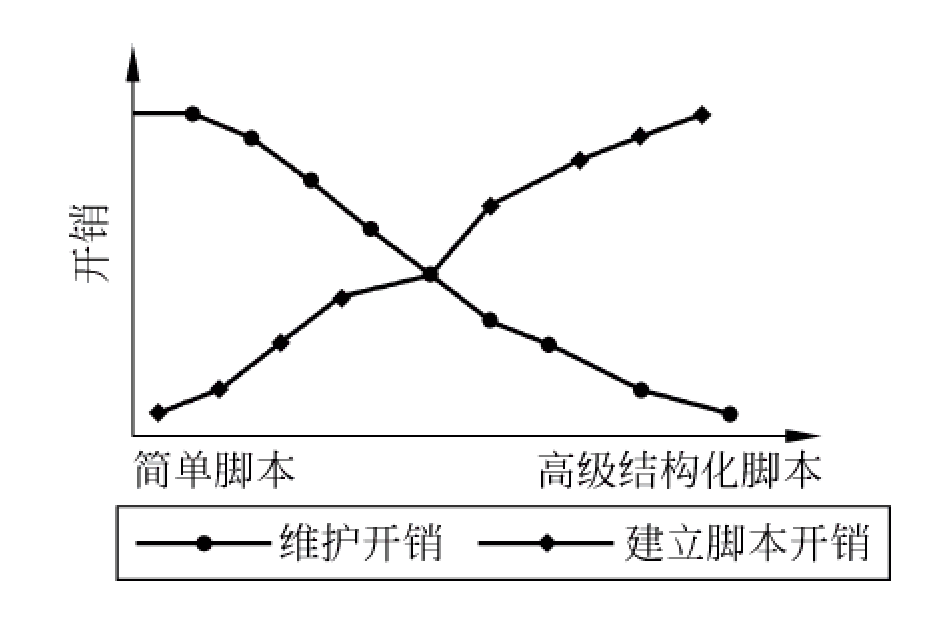


图 9-3 脚本开发成本与维护成本之间的关系

脚本可以通过录制测试的操作产生，然后再做修改，这样可以减少脚本编程的工作量。当然，也可以直接用脚本语言编写脚本。测试脚本中可以包含数据和指令，并包括一些信息：①同步（何时进行下一个输入）；②比较信息（比较什么、如何比较以及和谁比较）；③捕获何种屏幕数据及存储在何处；④从另一个数据源读取数据时从何处读取；⑤控制信息等。

脚本技术围绕着脚本的结构设计、实现测试用例，在建立脚本的代价和维护脚本的代价中得到平衡，并从中获得最大益处。各种脚本技术不是相互排斥的，而是相辅相成的，每种技术在支持脚本完成测试用例的时间和开销上各有长短。因此，使用哪种脚本技术并不重要，脚本所支持的实现测试用例体系的整体考虑才是最重要的。下面介绍几种脚本技术。

1. 线性脚本

线性脚本是录制手工执行的测试用例得到的脚本，它包含所有的击键操作（包括功能键、移动、输入数据的数字键等）。若只使用线性脚本技术，则所有录制的测试用例都可以得到完整的回放。对于线性脚本，也可以加入一些简单的指令，如时间等待、比较指令等，但是在进行重新回放时，新增加的脚本也应进行测试。应用和测试用例越复杂，这个回放过程就花费越多的时间。因此，线性脚本适合于简单的测试（如 Web 页面测试）、一次性测试，多数用于脚本的初始化（录制的脚本用于以后修改），或者用于演示等。

2. 结构化脚本

结构化脚本类似于结构化程序设计，具有各种逻辑结构，包括选择性结构、分支结构、循环迭代结构，而且具有函数调用功能。结构化脚本具有很好的可重用性、灵活性，所以结构化脚本易于维护，但是它也使得脚本变得更加复杂，而且测试数据仍然“捆绑”在脚本中。

3. 共享脚本

共享脚本是指某个脚本可以被多个测试用例使用，即脚本语言允许一个脚本调用另一个脚本。可以将线性脚本转换为共享脚本。这种技术的思路是产生一个执行某种任务的脚本，而不同的测试要重复这个任务，当要执行这个任务时只需在每个测试用例的适当地方调用这个脚本。这样将带来两个好处：①可以节省生成脚本（编写或录制指定的操作）的时间；②当重复任务发生变化时，只需要修改一处脚本。

共享脚本的优点是以较少的开销实现类似的测试，维护开销低于线性脚本，可以删除明显的重复，可以在共享脚本中添加更智能的功能。因此共享脚本技术比较适合小型系统或大型应用中只有一小部分需要测试。但是对于共享脚本来说，它需要我们跟踪更多的脚本、文档、名字以及存储，如果管理得不好，就很难找到适合的脚本；对于每个测试需要一个特定的测试

脚本，维护成本较高；而且共享脚本通常是针对被测软件的某一部分。

4. 数据驱动脚本

数据驱动脚本技术将测试输入存储在独立的(数据)文件中，而不是存储在脚本中，脚本中存放控制信息。执行测试时，是从文件中而不是直接从脚本中读取测试输入。这样的脚本可以针对不同的数据输入实现多个测试用例。例如，保险系统的一个测试是输入新保险策略的详情并验证数据库是否被正确修改。第二个测试除使用不同的保险策略外，进行相同的操作。因此需要相同的指令，但输入和期望输出不同(即描述不同的保险策略的值)。这两个测试可以使用一个测试脚本和一个数据文件。

使用数据驱动脚本技术，可以以较小的额外开销实现许多测试用例，因为需要做的工作只是为每个增加的测试用例指定一个新的输入数据集合，而不需要编写更多的脚本。另外，数据驱动脚本技术使得测试者易于处理数据文件，方便选择测试数据的格式和形式。但是，数据驱动脚本技术在初始建立时开销较大，需要专业编程支持以及必须易于管理。

5. 关键字驱动脚本

关键字驱动脚本技术实际上是数据驱动脚本的逻辑扩展。数据驱动技术的限制是每个测试用例执行的导航和操作必须一样，测试的逻辑"知识"建立在数据文件和控制脚本中，因此两者需要同步。

然而，脚本的一些智能活动不能移动到数据文件中。实际一点的方法是允许控制脚本支持广泛的测试用例，而这大大增加了数据文件的复杂性(因为数据文件此时要包含脚本指令)，这种代价很大。而且调试这种方法实现的自动测试用例是十分困难的。而关键字驱动脚本技术和数据驱动脚本技术相结合后，将数据文件变为测试用例的描述，用一系列关键字指定要执行任务的任务。控制脚本可以解释关键字，但是这是在控制脚本之外完成的。

关键字驱动脚本技术所需的脚本数量是随软件的规模而不是测试的数量而变化，因此，可以不用增加脚本的数量而实现很多的测试，只用替换基本的应用支持脚本，减少了脚本的维护开销；可以用与工具(及平台)无关的方法实现。

关键字驱动脚本技术主要应用于软件测试自动化的工程应用领域和数据库应用中。

实际工作中，在建立脚本时，通常会将几种技术结合起来应用，如数据驱动脚本技术和关键字驱动脚本技术经常是一起使用的。

脚本技术不仅可以用在功能测试上模拟用户的操作，然后进行比较，而且可以用在性能、负载测试上，虚拟用户同时进行相同或不同的操作，给系统或服务器足够的数据、操作，以检验系统或服务器的响应速度、数据吞吐能力等。

9.3.2 录制/回放技术

录制/回放是一种"黑盒"测试的自动化方法。录制/回放技术是在窗口系统的基于消息管理机制的基础上实现的，通过对被测系统的监控，获得被测系统的运行信息，把用户在被测系统上所做的操作和输入按照时间顺序全部记录在指定的测试脚本中。然后，所有的记录转换为一种脚本语言所描述的过程，以模拟用户的操作。

回放时，将脚本语言所描述的过程转换为屏幕上的操作，然后将被测系统的输出记录下来与预先给定的标准结果比较。这可以大大减轻"黑盒"测试的工作量，特别是在迭代开发的过程中，能够很好地进行回归测试。

1. 录制/回放技术分类

在 GUI 的自动化测试中录制/回放从技术上来说分为三种类型：①自动提供对用户手工

操作的录制/回放；②采用脚本语言模拟用户在被测程序上的操作；③是前两者的综合。

1）录制/回放用户手工的操作

这类录制/回放工具主要用于被测程序的回归测试中，它需要测试人员第一次手工对GUI进行操作，来完成测试脚本的录制过程。在录制过程中这类测试工具会首先记录被测程序GUI的组件层次结构和组件自身的信息，随后会截获测试人员在被测程序GUI组件上触发产生的事件，随后解析该事件，得到事件的各个参数，保存到测试脚本中，作为将来回放的依据。

测试脚本多以文本格式存放，主要记录两部分内容：①被测程序的GUI组件结构及组件自身的属性信息；②对测试人员手工操作步骤，也可以理解为对手工操作引发的事件的记录。

回放时，会使用某种脚本语言解析该记录文件，脚本语言可以分为外部类型和交互类型两大类：①外部类型的脚本只能进行整个被测程序的调用（如某个Java应用程序），它无法单独地针对被测程序内部的某个方法进行调用，因此外部类型的脚本语言经常是用来记录一系列的“黑盒”测试步骤；②交互类型的脚本语言可以访问到被测程序内部的所有属性和方法并与之交互，例如对于Java程序来说，交互型脚本语言应该能够访问Java的类、类成员函数以及成员变量。当脚本语言根据记录的事件信息重新构建了一个事件，随后会通过记录的组件信息找到作为该事件触发源的组件，然后将重新构建的事件操作再次应用到该组件上来完成回放功能。

此类录制/回放工具是最基本，同时也是最有效的，大部分为了完成回归测试的自动化工具都会选择这一种方式。另外，对于脚本录制得比较清晰、结构化比较好的测试脚本，还可以进行修改和添加，以更好地满足回归测试的需要。大多数这种类型的GUI录制/回放工具完成简单的“黑盒”录制/回放而不需要交互型脚本语言的介入，它们仅仅是简单地录制用户的操作步骤并回放，被测程序的运行时(Runtime)内部状态在这种方式下是不可见且无法修改的。

2）借助交互型脚本语言完成录制/回放的功能

交互型脚本语言是在一个Shell环境下，以一系列的命令的方式，对特定功能调用，一旦某个功能调用完毕，进程控制又会返回到Shell命令行，同时，当前程序的运行时上下文会被保存到脚本的Shell环境下，这样就可以在模拟用户与被测程序的GUI对象交互时完成“单步”操作。

采用交互型的脚本语言模拟被测运行程序单步执行的过程类似于程序开发中的“调试”过程，用户可以通过脚本对被测应用程序设置断点，使得被测应用程序在特定情况下的特定地方停止执行（这取决于脚本语言本身的功能），而这种“单步运行”优于调试的地方在于，采用脚本语言，用户可以动态地改变运行环境参数。

基于交互型脚本语言的录制/回放工具，它们并不是直接录制用户的操作步骤，而是编写出一批脚本以备回归测试进行回放。

3）基于操作和交互的录制/回放

这种类型的录制/回放工具集成了前两种类型的优点，既可以录制用户操作，又可以通过修改脚本完成断点设置、单步进行等操作，属于这种类型的录制/回放工具有代表性的是：IBM Rational公司的Robot和HP公司的QTP，这类测试工具功能强大，但属于商业录制/回放软件，需要不菲的资金投入。

2. 用户输入的捕获分级

在GUI自动化测试中，对于用户输入的捕获，GUI录制/回放工具将以三种级别捕获，即硬件级别、操作系统级别和进程级别。

1）硬件级别

硬件级别是最低的级别，主要是采用硬件完成对用户输入的录制/回放，比如鼠标的双击、键盘输入某个字符等。之所以采用硬件进行录制和回放，是出于安全的考虑，硬件设备可以捕获键盘敲击的键值，从而监控该计算机的使用，或者它可以加密键盘扫描码，以防止底层恶意代码对计算机的监听。

2）操作系统级别

在操作系统级别下，录制/回放功能采用软件完成。软件实现主要是使用本机代码(Native Code)方式调用的系统 API。操作系统级别的录制/回放工具主要用于在系统级别对键盘或鼠标的输入进行监控，由于每个 GUI 组件都存在一个在系统级别里可被识别的 Global ID，所以系统级别的录制/回放工具可以从系统级别的输入队列中提取出所需的 GUI 组件的输入。但是，在操作系统级别下，由于是调用系统 API 完成录制/回放，工具无法做到完全识别 Java 对象，因为 Java 对象必须在 Java 虚拟机(JVM)之上被识别和使用；并且使用本机，API 的调用与操作系统平台相关，使得这种级别的录制/回放工具无法做到很轻巧。

3）进程级别

进程级别的含义是说录制/回放工具是对特定的进程进行监控，一般来说，这个级别的录制/回放工具可以采用 C++或者 Java 语言实现，并且通过在测试工具中对不同语言编写的目标程序加挂针对该语言的包，就可以监控大多数语言编写的目标程序，其中加挂的不同语言包主要是针对这种语言编写的 GUI 组件的识别信息。特殊地，对于 Java 编写的目标应用程序，同样采用 Java 编写的进程级录制/回放工具来监控会比较简单有效，因为这样目标程序和测试工具都在 JVM 解释下执行，测试工具可以通过 Java 语言中的反射机制识别到目标应用程序内部的 Java 类、成员函数和成员变量，扩展第一类 GUI 录制/回放工具的功能。

3. GUI 对象识别技术

在基于 GUI 的软件系统中，所有的功能都是通过对界面上可见对象的操作来完成。对这些 GUI 对象的识别是基于 GUI 的自动化测试中的重要问题。

一般的方法是通过 GUI 对象控件的属性和属性值（如 id、text 等）进行对象识别。Maryland 大学的 Atif Memon 提出了一种 GUl 建模方法，GUI 被表示为对象集及其属性集、状态、事件、组件等构成的一个模型。目前主流的自动化功能测试工具，如 IBM Robot、HP QTP 等都是通过对象属性进行对象识别的。除此之外，也有用 GUl 的图像来进行对象识别的。

9.3.3 基于数据驱动的自动化测试技术

在经历了录制/回放、编写脚本、结构化脚本等自动化测试技术发展后，一种新的自动化测试技术逐渐被测试人员提了出来，那就是基于数据驱动的自动化测试技术。

数据驱动的自动化测试技术是测试从数据文件（数据池、ODBC 源、CVS 文件、Excel 文件、DAO 对象、ADO 对象等）中读取输入和输出数值并载入到捕获的或者手工编码的脚本中变量的一种框架。数据驱动需要的代码很少，但能产生大量的测试用例。在这个框架中，变量不仅用来存放输入值，还用来存放输出的验证值。整个执行过程中，测试脚本来读取数据文件，记载测试状态和信息。

1. 数据驱动的自动化测试技术的优点

基于数据驱动的自动化测试技术有以下优点：

(1) 相比前几种自动化测试技术，减少了测试脚本的维护。

(2) 使用简单的输入文件，测试数据有较高的可维护性。

(3) 用输入数据控制测试的执行。

(4) 当增加额外的测试数据时，不必修改测试脚本。

(5) 自动化测试开发人员创建数据驱动测试过程，测试人员创建数据文件，提高了测试效率。

2. 数据驱动的自动化测试技术的缺点

基于数据驱动的自动化测试技术存在以下缺点：

(1) 脚本维护比较困难。当被测软件的功能出现增加或变化后，脚本需要及时更新。

(2) 数据文件维护烦琐。测试人员不仅要维护不同测试用例需要的数据文件，而且同一个测试用例也可能有多个测试文件需要维护。

(3) 测试人员同样需要对自动化测试工具里的脚本很熟悉。

9.3.4 基于关键字驱动的自动化测试技术

关键字驱动的自动化测试技术是现在自动化测试中被广泛使用的一种自动化测试技术，也是比较成熟的一种自动化测试技术。关键字驱动是一种独立于应用程序的自动化测试框架。这种框架要求开发用来运行的自动化工具，使用与程序的测试脚本代码相独立的数据表和关键字作为框架的输入。关键字驱动的自动化测试框架建立在数据驱动手段之上，测试表中既包含了关键词也包含了测试数据。测试用例被开发成使用关键字的数据表，它们独立于执行测试的自动化工具。关键字驱动的自动化测试是对数据驱动的自动化测试的有效改进和补充。

关键字驱动的自动化测试框架是一种截然不同的思想，它把传统测试脚本中变化的与不变的东西进行了分离，这种分离使得分工更明确，并且避免了它们相互之间的影响。这种模型的开发和实现与传统的测试流程相比可能是困难的、耗时的，它需要将具体测试和自动化工具以及应用程序本身的变化完全隔离开来。但是这样的投资是一次性的，一旦开发结束并投入使用，它给我们带来的效益是巨大的，是自动化测试框架中最容易维护和使用的，而且可以反复运用于各种应用中，长期发挥作用。此外，现在已经有一些符合需求的商业化产品可供使用，减少了实现这种框架的困难。

1. 关键字驱动的自动化测试技术的优点

基于关键字驱动的自动化测试技术主要有以下优点：

(1) 支持测试多样化。由于可以对不同序列或多个路径的关键字序列进行测试，因此测试更加多样化，测试覆盖更加全面。

(2) 测试易于维护。原因是测试用例与测试脚本实现了分离。

(3) 测试人员不需要具有很强的技术背景。由于测试人员不再接触测试脚本，只需要设计基于关键字的测试用例，因此自动化测试的门槛大大降低。

(4) 投入产出高。对基于关键字驱动的自功化测试工具的开发是一次性的，而对它的使用却是长期的，因此投资的效益巨大。

2. 关键字驱动的自动化测试技术的缺点

基于关键字驱动的自动化测试技术也有它的缺点：

(1) 有可能对基于关键字驱动的自动化测试工具的开发投入巨大，造成人力和时间资源的短缺。

(2) 对于小型项目，实行基于关键字驱动开发是不合适的。

(3) 关键字需要做到很好的维护。当基于关键字驱动的自动化测试工具被长期使用时,关键字可能也是一个庞大的数目,大大降低了测试工具的易用性。

总之,关键字驱动技术实现了脚本、数据和业务的分离,提高了可复用性和健壮性。

9.3.5 基于测试预期的结果分析比较技术

测试结果分析比较是检验软件是否产生了正确输出的过程,是通过在测试的实际输出与期望输出之间完成一次或多次比较来实现的。一些测试用例只要求一种比较来验证软件的输出。例如,一个验证用户登录的测试用例,而另一些测试用例则会要求进行多种比较。如添加一条记录到系统中,不仅仅要检验在屏幕上的返回信息,还要去验证数据库中是否有相应数据被添加。

一个测试用例的两个最主要的组成部分就是测试步骤和预期结果。同样,一个健壮的自动化测试集不仅仅包括自动化的测试输入和执行,还要包括高效的自动化测试结果比较分析。自动化测试的一个目的就是减少人工在测试过程中的参与程度,同时可以以较小的成本运行大量的测试用例。如果自动化测试没有一个可靠的测试结果比较分析机制,这样的自动化测试解决方案必定是纸上谈兵,没有任何的实际运用价值。一个自动化测试集在一个晚上可能执行数千个测试用例,如果由测试人员手工比较和分析这数千个测试用例产生的测试结果,这个过程将是相当烦琐和低效的。

1. 被测系统输出结果分类

下面将针对不同类型的系统输出结果来讨论不同的分析比较办法。

1) 基于文件形式的系统输出

文件包括文本文件和非文本文件。对于文件形式的输出一般要借助于一些外部工具的帮助。这些外部工具可能是自动化测试工具附带的功能,也可能是自动化测试工程师自行编写的一些比较工具。但无论使用何种工具,文件形式比较的基本原理是利用工具读取实际文件和目标文件内的数据,然后进行逐行字符比较。

对于一些非文本的文件,通常关心的是这些文件内的数据而并非文件本身。所以同样可以使用文件比较器来对非文本文件的实际输出文件和目标文件进行逐字符比较。

2) 基于用户界面的系统输出

根据被测系统的性质,需要比较的系统界面输出一般为三种:命令行界面字符输出、图形化用户界面(GUI)的输出和图形图像输出。

(1) 命令行界面字符输出。如果测试基于字符的应用程序,那么比较屏幕输出时必须注意的是文本、特殊字符以及诸如粗体、闪烁和反白显示等显示属性。因为屏幕上的每个字符都用行列号来定址,所以通常指定要比较的字符位置是相当容易的。

(2) 图形化用户界面的输出。包含 GUI 的应用程序可能会有比基于字符的应用程序更广泛的输出类型。对绝大多数用户来说,这些输出包括窗口、图标、菜单、对话框、按钮、复选框、文本框这样的对象以及它们的属性。在对某一个 GUI 对象进行比较之前,确切知道它是一个什么样的对象?它能否方便地进行比较?主流的自动化测试工具(如 HP QTP)会通过一些对象映射的技术来得知被比较对象的类型,如一个标准按钮或者一个文本框,然后去访问这个对象的一些标准属性来实现比较。

(3) 图形图像输出。图形图像由被称为像素的点的精细阵列构成。用于生成图像的像素越多,所生成的图像细节就越精细。在必须要校验图形图像时,其实现的方法是通过比较实际图像和目标图像的位图文件来找出其中的差异。但是,这样的位图比较是一件极其麻烦的事

情,因为对于这样的比较方法来说,只要有一个位的数据不同就会出现不匹配而视为不同的位图文件,尽管这两个图像文件即使从专业的角度去观察也无法发现任何差异。所以一些位图比较器为了忽略无法察觉的差异,允许用户指定一个容错度。总而言之,位图比较是尽量要避免的比较方法,因为位图过于敏感,在实际测试中实用价值不大。

3)基于数据库的系统输出

自动化分析比较的一个优点就是,可以很方便地实现实时的数据库比较。在某个界面操作以后,测试脚本可以立即通过查询数据库的方法来证实该项操作是否引起了正确的数据库操作。这在手动测试结果比较中是一件费时费力的工作,但在自动化测试结果比较中却是最常用的也最有效的比较方法。

主流的自动化测试工具都支持通过 ODBC 或者其他的数据库连接引擎来实现对数据库的访问,包括查询、添加、删除记录等。这样实现对数据库系统输出的比较就仅仅是一些技术层面的问题了。

2. 结果分析比较方法

自动测试时,预期输出是事先定义的。在自动测试运行脚本过程中,将捕获的结果和预先准备的输出进行比较,从而确定测试用例是否通过,或分析预期结果与实际结果是否一致,进而分析结果不一致的原因。所以基于测试预期结果的分析比较在软件测试自动化中就非常重要。基于测试预期结果的分析比较包含以下几类方法:①动态比较,是在测试过程中进行比较;②静态比较,在测试过程中并不作比较,而是将结果存入数据库或文件中,然后通过另外一个单独的工具来进行结果比较;③简单比较,简单比较要求实际结果和期望结果完全相同;④复杂比较,是一种智能比较,允许实际结果和期望结果有一定的差异,智能比较需要使用屏蔽的搜索技术,来排除输出中预期会出现差异部分,忽略特定的差异;⑤敏感性测试比较,该比较方法要求比较尽可能多的信息,如在执行测试用例的每一步就比较整个屏幕的信息,屏幕输出中或多或少的变化就可能导致不匹配,而表明此测试用例失败;⑥健壮性测试,只比较最少量、最需要的信息,如屏幕的最后输出。

在比较过程中经常要用到比较过滤器。比较过滤器就是在对实际输出结果和期望输出结果进行预先处理,执行过滤任务之后,再进行比较。这样可以使比较标准化,测试结果可靠。

在上述介绍的几类分析比较方法中,最重要的是动态比较,而动态比较分两种情形。

1)运行时比较

运行时比较是指测试脚本在运行的同时即进行相应的比较。例如,当测试用例是向系统添加一个用户,那么一个可能的运行时比较就是在测试脚本完成相应的测试步骤后,结果分析比较模块会去捕获屏幕相应位置的输出信息。这些输出信息可能是“错误!用户已存在!”,或者是“用户添加成功!”。当将这些系统的输出信息捕获以后,自动化测试脚本会和该测试用例的期待结果作比较,从而得出该测试的结果。

另一种情况是,系统的输出结果和预期结果可能会有一定的误差,这些误差是在测试的许可范围之内。例如,测试脚本捕获到的系统输出可能是“用户添加已成功!”,然而期望结果是“用户添加成功”。这样的情况比较普遍,而且往往是一些自动化测试脚本健壮性不强的原因之一。一般的处理方法是测试脚本不是对两个结果(实际结果和预期结果)作简单的完全匹配,而是作一个关键字匹配。像前面那个例子,测试脚本会在实际结果中作两个关键字(添加,成功)的匹配,如果这两个关键字都能在实际结果中正确匹配,测试脚本将认为此次测试是成功的。这样就解决了一些界面微小变动对测试脚本健壮性的影响。

几乎所有的屏幕输出比较都是运行时比较。原因也比较容易理解,因为几乎所有的屏幕

输出都是暂时性的输出，这些输出如果没有被及时对比，其包含的信息就可能丢失。

2）运行后比较

运行后比较是自动化测试中另一种重要的比较方法。和运行时比较不同，其对结果的比较和分析是在该测试用例完全执行完成以后才发生。运行后比较主要用于比较发送到屏幕以外的输出，如创建的文件和数据库中已更新的内容。

可以主动地或被动地进行运行后比较。如果仅仅试图看测试用例执行后剩下了什么信息、可供比较分析，那么这就是一个被动的运行后比较。例如，前面举的那个添加和删除用户的例子，当整个测试用例被执行后，什么也没有剩下。这种情况就需要运行时比较。但是，也可以使用主动的运行后比较。

如果在运行测试用例时，有意保存感兴趣的特定结果，为了以后特殊的目的比较这些结果，那么这就是主动的运行后比较。在前面的例子中，如果在添加用户的测试步骤完成以后，测试脚本发出指令去捕获屏幕的特定输出部分，并保存在一个文件中，然后再删除用户时，测试脚本再发出指令去捕获屏幕的特定输出部分，并保存在同一个或一个新文件中。这样，主动的运行后比较和运行时比较的效果完全相同。

9.4　自动化测试工具应用举例

自动化测试需要不同类型的自动化测试工具进行支持。现在市面上有商用软件测试工具，包括 IBM Rational Robot、HP QTP 等，还有一些开发源代码的测试工具，如 Ant、JUnit、JProbe 和 Cactus 等。

9.4.1　测试中常用的自动化测试工具

下面我们根据在测试过程中需要使用哪些测试工具来介绍。

1. 测试管理工具

测试管理工具对测试需求、测试计划、测试用例、测试实施进行管理，并且测试管理工具还包括对缺陷的跟踪管理。测试管理工具能让测试人员、开发人员或其他的 IT 人员通过一个中央知识库，在不同地方就能交互信息。代表性的有 IBM Rational Test Manager、HP Quality Center 等软件。

2. "白盒"测试工具

"白盒"测试工具一般是针对代码的内部逻辑流程和结果进行测试，测试中发现的缺陷可以定位到代码级。根据测试工具原理的不同，又可以分为静态测试工具和动态测试工具。

1）静态测试工具

静态测试工具直接对代码进行分析，无须编译运行。静态测试工具用于两个方面：①代码评审，一般是对代码进行语法扫描，找出不符合编码规范的地方；②静态结构分析，根据代码流程图计算复杂度评价代码设计，生成模块的调用关系图等。

静态测试工具的代表有 IBM Rational Logiscope 软件、GIMPEL PC-Lint 软件等。

2）动态测试工具

动态测试工具需要运行代码，一般采用"插桩"的方式，向代码生成的可执行文件中插入一些监测代码，用来统计程序运行时的数据及覆盖率。

动态测试工具的代表有 IBM Rational Logiscope、Parasoft C++test 等。

3. "黑盒"测试工具

"黑盒"测试工具又称功能测试工具，常通过自动录制、检测和回放用户的应用操作，将被

测系统的输出记录同预先给定的标准结果比较。功能测试工具能够有效地帮助测试人员对复杂的企业级应用的不同发布版本的功能进行测试,提高测试人员的工作效率和质量。其主要目的是检测应用程序是否能够达到预期的功能并正常运行。主要有 GUI 测试驱动和捕获/回放工具和非 GUI 测试驱动工具。功能测试工具的代表有 IBM Rational Robot 和 HP Quick Test Professional(QTP)。

4. 性能测试工具

性能测试工具的主要目的是度量应用系统的可扩展性和性能,是一种预测系统行为和性能的自动化测试工具。在实施并发负载过程中,通过实时性能监测来确认和查找问题,并针对所发现问题对系统性能进行优化,确保应用的成功部署,负载压力测试工具能够对整个企业架构进行测试,通过这些测试,企业能最大限度地缩短测试时间,优化性能和加速应用系统的发布周期。性能测试工具的代表有 HP LoadRunner、Segue SilkPerfomer,以及开源的 Apache JMeter。

5. 测试辅助工具

测试辅助工具本身并不执行测试,例如它们可以生成测试数据,为测试提供数据准备。

针对不同类型的测试选择专门的自动化测试工具具有重要意义,如一个工具用于脚本测试和界面控制,一个工具用于向服务器发送直接请求,一个工具用于验证应用程序接口(API)的功能,一个工具用于验证代码是否标准,以及其他一些认为必要或需要的测试工具。

9.4.2 基于 STAF/STAX 的自动化测试框架

如今软件开发依赖于团队的开发和测试。对于部署和测试人员来说,如何从集中的代码管理工具来获取源代码或者代码的编译包并且自动部署和测试变得非常重要。为此,我们对自动化测试框架 STAF/STAX 进行简要的介绍。

STAF(Software Test Automation Framework)来源于 IBM,是开源、跨平台、支持多语言并且基于可重用的组件来构建的自动化测试框架。它为自动化测试建立了基础,并且提供了一种可插拔的机制支持不同的平台和语言。STAF 采用点对点的实现机制,被用来减轻自动化测试的工作负担,加快自动化测试的进程。在 STAF 的环境中,所有的机器都是对等的,没有客户端和服务器的区分。

简而言之,STAF 就是一个在不同机器、不同的操作系统之间提供一个通信的平台。STAF 利用其开源的优势,经历了最近几年的发展,已经越来越成熟,其特点鲜明:①简单易用,可快速搭建一个跨平台自动化测试环境;②开源,易于扩展,用户可以方便地在 STAF 中创建一个新的服务,环境要求低;③支持多种平台多种操作系统;④支持多种语言,可以在 Java,Linux Shell,C/C++,Python,Perl 等各种语言中调用。

Software Test Automation eXecution Engine(STAX)是基于 STAF 的执行引擎。它在 STAF 的基础上,帮助用户实现测试用例的分发、部署、执行以及结果分析。STAX 使用了三种技术:STAF,XML 和 Python。简单来说,STAX 在 STAF 之上提供了一些接口,方便用户来操纵 STAF 进行自动化测试的实现。

下面详细介绍 STAF 一些概念。

1. 原理

假设我们有 2 台机器 A 和 B,A 是主控机,如图 9-4 所示。

由图 9-4 可以看出,机器 A 和 B 都安装了 STAF,并且相互配置了信任关系。那么用户就可以通过在机器 A 上调用 STAF 的 Service 来实现与机器 B 的相互通信。如传输文件、操作

机器 B 等。所以,STAF 的作用实际上就是提供机器之间的通信通道并提供基于这个通道的基础服务。

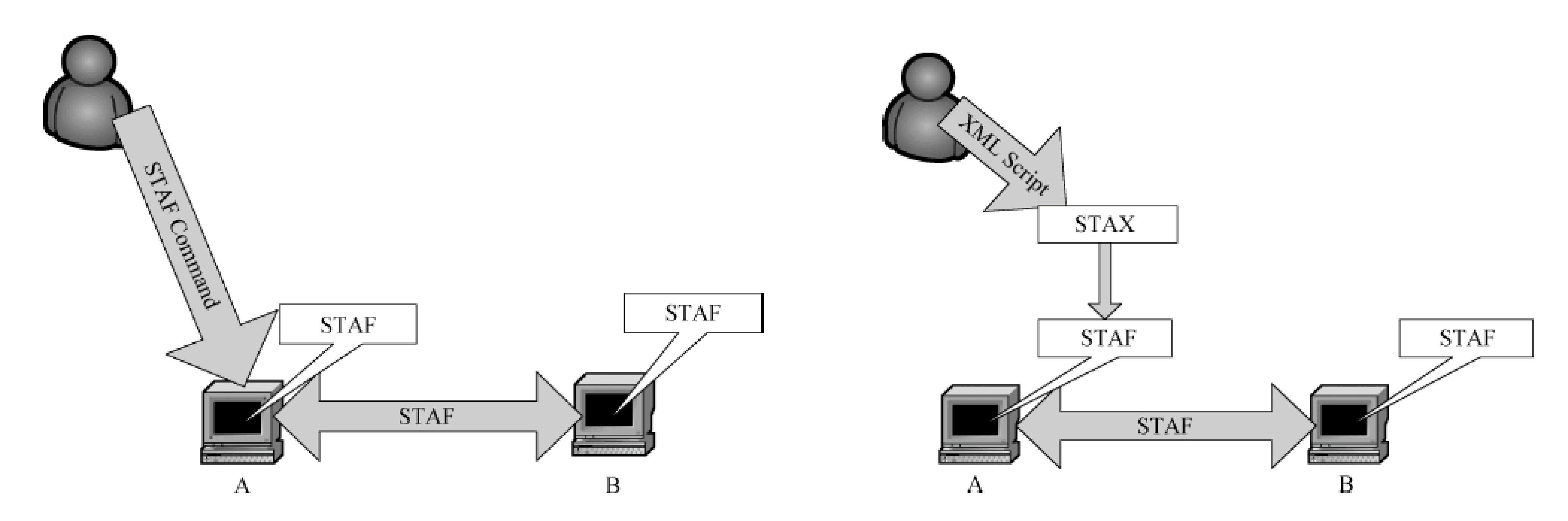


图 9-4　2 台机器上 STAF 相互通信

图 9-5　基于 STAF/STAX 的通信

2. 多层封装概念

通过上面的介绍,我们了解了 STAF 是一个可重用并对各种应用提供广泛支持的基础框架。于是,就有了根据不同应用产生的不同封装。比如我们构造了一个复杂的分布式测试环境,我们可以单独通过 STAF 将我们的测试任务分发到不同的测试环境去执行,但如果测试任务太多,并且是不断添加新的任务进来,单独依靠 STAF 就不利于测试任务的执行、管理与维护了,从而也就引入 STAX 的概念,如图 9-5 所示。

由图 9-5 可以看出,机器 A 安装了 STAF/STAX,那么用户可以把一批 STAF Service 调用通过 XML 格式写在一个文档里,然后由 STAX 调用这个文档并翻译成 STAF Service 通过 STAF 传递给机器 B 执行。

另外,在实际测试中,有些工作光靠 STAF 是完不成的。我们往往需要利用一些工具来完成自动化测试,如 Robot。这就需要提供一个自动化测试工具与 STAF 框架之间的接口。SAFS 就是干这个的。在 SAFS 里,引入了关键字驱动(Keyword-Driven)和数据驱动(Data-Driven)的概念。通俗点说,SAFS 就是为了给第三方软件提供支持,将一些常用动作(Action)进行封装。并提供对大量数据操作的接口。

目前基于 SAFS 的成熟应用接口是 RRAFS(Rational Robot Automation Framework Support)。现在已经支持 IBM Rational Robot,IBM Rational Functional Tester,SAFS Image-Based Testing 及 HP LoadRunner 等第三方软件,如图 9-6 所示。

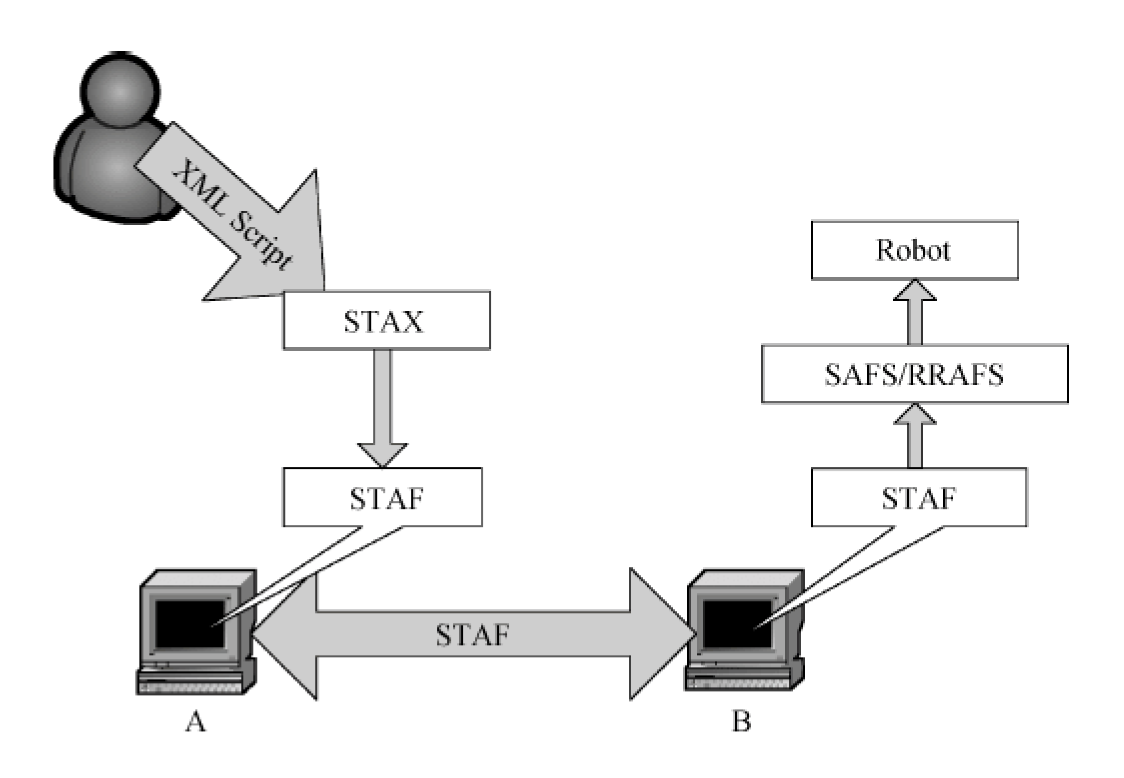


图 9-6　集成有第三方测试工具的 STAF/STAX 通信

3. Services(服务)

STAF基于可重用的组件来构建自动化测试框架,这些可重用的组件就是Services(服务)。STAF中所有的组件都是服务。服务是一系列功能的集合。STAF本身是一个后台程序(STAFProc),提供一种轻量级的分发机制,负责把请求转发给这些服务。

STAF中的服务分为internal(内部服务)和external(外部服务)两种:①内部服务被集成进STAFProc中,提供一些关键性的功能,如数据管理和同步;②外部服务由STAFProc动态装入,通过共享库(shared libraries)来访问。

STAF提供了以下几种常用服务。

(1) 程序调用服务(Process Service):内部服务,利用此服务,STAF可调用外部程序。

(2) 文件系统服务(FileSystem Service):内部服务,利用此服务,STAF可以对文件系统进行操作,如复制、删除、查看等操作。

(3) 日志服务(Log Service):外部服务,帮助用户进行日志的记录和查看。

(4) 资源池服务(ResPool Service):外部服务,提供了对于资源池的管理和操作,如查看、创建和删除操作。

(5) 监控服务(Monitor Service):外部服务,提供对于STAF运行时的监控功能。

(6) 信号量服务(Sem Service):内部服务,提供了两种信号量的操作,mutex和event。

(7) 压缩服务(Zip Service):外部服务,提供了压缩和解压的功能。

(8) Ping服务(Ping Service):内部服务,类似于操作系统的Ping功能,用于检测远程的STAF是否运行。

(9) 变量服务(Var Service):内部服务,提供对于系统或者用户级别的环境变量的操作。

(10) 其他:STAF还提供了延迟服务(Delay Service)、帮助服务(Help Service)、跟踪服务(Trace Service)等服务。

4. 请求-响应格式

每个服务都定义了它能接受的请求格式。STAF通过请求来调用服务的功能,每个请求都以字符串的形式发送,这样可以保证STAF能够跨平台地运行。每个请求都有三个参数,以系统-服务-参数的形式出现。第一个参数表示此请求需要被发送到的STAF系统,这个参数被STAFProc解析以便确定请求应该被本地处理还是发送到其他的STAF系统。当这个请求被发送到需要处理的STAF系统后,STAFProc解析第二个参数来判断哪个服务会被调用。最后,STAFProc会把第三个参数转发给需要调用的服务,服务处理这个请求。

当处理完请求后,服务会返回两种数据:返回码和特定于请求的信息。返回码表示服务处理的结果。特定于请求的信息表示服务返回的具体数据,如果请求成功返回,这些信息将包括这次请求所请求的数据,如果请求出现错误,这些信息将包含额外的诊断信息。

完全使用字符串作为请求响应格式可以简化STAF的很多方面,包括与其他语言的接口、服务之间的通信、跨平台的操作等。其他语言只需要通过一个接口STAFSubmit()来请求STAF的服务,并且只需传递三个字符串参数。服务之间也只需要通过字符串发送接收请求。

5. STAX

STAX是基于STAF的执行引擎,它提供了一种XML格式的工作流语言。用户可以编写XML的脚本文件来通过STAX调用STAF的服务以完成自动化测试。用户可以不需要和编程语言打交道就可以开发出自己的自动化测试环境。STAX提供以下功能:支持并行运行,用户自定义的运行控制粒度,嵌套测试用例,控制运行时间,支持现有的Java和Python模块等。STAX还提供了一个图形化的监控工具,通过这个工具,用户可以清晰地看出测试运行

的位置、状态和出错信息等。

6. STAF 基础用法及简单配置

1）基础用法

```
STAF [ - verbose] <Endpoint> <Service> <Request>
```

- -verbose 指定输出采用 verbose 模式，verbose 是一种数据结构。
- <Endpoint> 操作对象 IP。指定目标 STAF 系统，由 STAFProc 解析以便确定是在本地处理还是发送到远端 STAF 系统。本机的话可直接写 local。
- <Service> 指定要调用的服务。
- <Request> 调用服务所需的参数。

当服务处理结束将返回两类数据：①表示服务处理结果的返回码(即服务是成功还是失败)；②该服务返回的特定数据。

2）STAF 的简单安装配置

我们假设有 2 台机器，机器 A 是 Windows 机器，IP 是 10.2.5.100，机器 B 是 Linux 机器，IP 是 10.2.5.101。

(1) 安装包下载

从 http://staf.software.informer.com/download 下载所需安装包，有 Windows、Linux、Solaris、Mac 等不同平台不同的版本的安装包。如果机器上未安装 JDK，可以下载含 JDK 的安装包。

(2) Windows 下的安装

① 在 Windows 平台上，双击 STAF 的安装包，按照提示进行安装即可。

② 修改 STAF 安装目录\STAF\bin 中的 STAF.cfg 文件，配置机器间的信任关系。把 Linux 机器的 IP 配置进 config 文件。如下所示：

```
# Set default local trust
trust machine local://local level 5
trust machine 10.2.5.101 level 5
```

③ 重新启动 STAFProc。

④ 在 CMD 下键入下列命令来确认 STAF 已经安装正确：

```
staf local service list
```

(3) Linux 下的安装

假如我们选定 Linux AS release 3，那么测试机安装前请先下载 libstdc++-3.4.5-2.i386.rpm 并安装。否则 STAF 起不来。下载地址：

http://rpm.pbone.net/index.php3/stat/4/idpl/2790009/com/libstdc++-3.4.5-2.i386.rpm.html

Linux 下的 STAF 有两种安装模式：①直接下载 bin 包进行安装；②下载 tar 包进行安装。以 tar 包安装为例。

① 解压安装包：tar xzvf STAF331-linux.tar.gz。

② 解压后会出现一个 STAF 文件夹，进入该文件夹./STAFInst 进行安装，默认会被安装到/usr/local/staf 下。

③ 配置环境变量，将下列代码加入到要运行 STAF 的用户名目录下的.bash_profile 文件里：

```
PATH = /usr/local/staf/bin: $ PATH
export PATH
LD_LIBRARY_PATH = /usr/local/staf/lib
export LD_LIBRARY_PATH
CLASSPATH = /usr/local/staf/lib/JSTAF.jar:/usr/local/staf/samples/demo/STAFDemo.jar
export CLASSPATH
STAFCONVDIR = /usr/local/staf/codepage
export STAFCONVDIR
STAFCODEPAGE = LATIN_1
export STAFCODEPAGE
nohup /usr/local/staf/bin/STAFProc >
/usr/local/staf/stafproc.out
```

④ 修改 STAF 安装目录/usr/local/staf/bin 下的 STAF.cfg 文件，配置机器间的信任关系。把 Windows 机器的 IP 配置进 config 文件。如下所示：

```
# Set default local trust
trust machine local://local level 5
trust machine 10.2.5.100 level 5
```

⑤ 启动 STAFProc：/usr/local/staf/bin/STAFProc&

⑥ 确认 STAF 已经安装正确：

```
staf local service list
```

⑦ 测试机器 A、B 的 STAF 是否配置成功，可在机器 B 上输入以下命令，执行后机器 A 会启动记事本：

```
STAF 10.2.5.100 PROCESS START COMMAND notepad
```

习题

1. 什么是手工测试？什么是软件测试自动化？软件测试自动化一般用于何种场合？
2. 什么是软件测试自动化框架？常用的自动化框架有哪些？如何应用这些框架？
3. 软件测试自动化常用技术有哪些？如何使用这些技术？
4. 在 Windows/Linux 上搭建 STAF 环境，并开展自动化测试。

第10章 软件可靠性测试

前面我们学习了各种软件测试方法和技术，但通过它们只能尽量发现错误、减少错误，却不可能彻底消除错误，无法实现真正意义上的无错软件。于是，人们借鉴硬件可靠性理论，提出了"软件可靠性"概念，不再追求软件无错，而以统计的观点去判断软件满足用户使用要求的程度。

从前面我们知道，软件可靠性是软件质量特性中重要的固有特性和关键因素，它反映了用户的质量观点。

软件可靠性的定义与硬件可靠性的定义相似，即软件按规定的条件，在规定的时间内运行而不发生故障的能力。软件可靠性不但与软件中存在的缺陷有关，而且与系统输入和系统使用有关。

从定义上看，评价或测试一个软件的可靠性，最主要的是要确定系统怎样输入、如何使用一整套的与软件可靠性相关的方法和策略。这样，我们可以给软件可靠性测试下定义，即软件可靠性测试是指为了达到或验证用户对软件的可靠性要求而对软件进行的测试，是一种有效的软件测试和软件可靠性评价技术。尽管软件可靠性测试也不能保证软件中残存的错误数达到最少，但经过软件可靠性测试可以保证软件的可靠性达到较高的要求。

软件可靠性测试是在软件的预期使用环境中，为了最终评价软件系统的可靠性而运用建模、统计、试验、分析、评价等一系列手段对软件系统实施的一种测试。它应该是面向故障的测试，以用户将要使用的方式来进行，每一次测试代表用户将要完成的一组操作，使测试成为最终产品使用的预演。

软件可靠性测试是获取软件可靠性估算数据的重要手段，它通过发现软件系统可靠性缺陷，为软件的使用和维护提供数据，确认软件是否达到可靠性的定量要求。目前软件可靠性测试用得最多的方法是类似于硬件操作剖面上的统计测试方法，即基于被测软件操作剖面的统计测试方法。这是最为传统、经典的软件可靠性测试方法。通过这种方法，能够实现软件可靠性的定量评估，从而有效地保障和提高软件的质量。

10.1 操作剖面与统计测试

操作剖面(Operational Profile)是指系统测试数据输入域以及各种输入数据的组合使用概率。软件的操作剖面"是指软件对使用条件的定义，即软件的输入值用其按时间的分布或按它们在可能输入范围内的出现概率的分布来定义"。粗略地说，操作剖面是用来描述软件的实际使用情况的。操作剖面是否能代表、刻画软件的实际使用取决于可靠性工程人员对软件的系统模式、功能、任务需求及相应的输入激励的分析，取决于他们对用户使用这些系统模式、功能、任务的概率的了解。操作剖面构造的质量将对测试、分析的结果是否可信产生最直接的

影响。

这种测试方法是在对软件的实际使用情况进行统计的基础上建立软件的操作剖面，并采用统计测试的方法进行测试。用这种方法所获得的测试数据与软件的实际运行数据比较接近，可直接用于软件可靠性估算。

操作剖面的建立可以与软件开发并行进行，这样也可以缩短软件开发和发布所需的时间。通过建立操作剖面可以量化测试代价，为软件可靠性估算提供依据。

统计测试(Statistical Testing)是指通过对输入统计分布进行分析来构造测试用例的一种测试设计方法。统计测试标识出频繁执行的部分，并相应地调整测试策略，针对这些频繁执行的部分进行详尽的测试。通过提高关键模块的安全性和可靠性，来提高整个系统的安全性和可靠性，以提高测试的性价比。进行统计测试的前提条件就是生成如实反映系统使用情况的使用模型。以往使用模型的建立主要是通过预测和估计得出的，不能如实地反映系统的真实情况。

10.2 基于操作剖面的软件可靠性测试

软件可靠性测试与传统意义上的软件测试不同，软件可靠性测试是统计测试思想在软件可靠性度量上的应用，要求测试按照软件的操作剖面进行，测试结果使用软件可靠性模型进行评价，其中操作剖面的构造是进行软件可靠性测试的基础。

通过软件可靠性测试可以有效地发现程序中影响软件可靠性的缺陷，从而实现可靠性增长。由于软件可靠性很大程度上依赖条件的变化，特别是软件操作剖面的输入分布，因此，我们可以基于操作剖面按照概率统计方法来设计输入的分布，对软件进行可靠性测试。一般先暴露在使用中发生概率高的缺陷，然后是发生概率低的缺陷。而高发生概率的缺陷是影响产品可靠性的主要缺陷，通过排除这些缺陷可以有效地实现软件可靠性的增长。

另外，通过对软件可靠性测试中观测到的失效情况进行分析，可以验证软件可靠性的定量要求是否得到满足。

最后，通过对软件可靠性测试中观测到的失效数据进行分析，可以评估当前软件可靠性的水平，预测未来可能达到的水平，从而为开发管理提供决策依据。软件可靠性测试中暴露的缺陷既可以是影响功能需求的缺陷，也可以是影响性能需求的缺陷。软件可靠性测试方法从概念上讲是一种黑盒测试方法，因为它是面向需求、面向使用的测试，它不需要了解程序的结构以及如何实现等问题。

软件可靠性测试通常是在系统测试、验收、交付阶段进行，它主要是在实验室内仿真环境下进行，也可以根据需要和可能在用户现场进行。

10.2.1 基于操作剖面的统计测试

在基于操作剖面的统计测试中，首先需要建立描述软件操作情况的操作剖面，然后利用操作剖面从所有软件操作中获得统计上正确的采样。软件的操作剖面，或称为软件的使用模型，刻画了在预期环境中软件各种预期操作的统计分布。在基于软件操作剖面的统计测试中，能够保证在测试过程中较早地发现那些在软件操作使用中最经常发生的失效。

对于有高可靠性要求的软件，在发布并投入实际使用之前，需要花费一定的时间和经费进行软件测试。一般来说，软件测试的时间和预算都是有限的，因此如何使测试预算最有效地发挥作用就显得非常重要。尽管人们对于各种测试方法的效果看法不一，但功能测试需要覆盖

软件所有的用户功能则是一致的，其用到的方法大致包括：①输入域划分方法，将软件输入域进行同类划分，在每类中选择测试输入进行测试；②统计测试方法，按照软件的使用模型选择测试输入，进行随机测试。

软件系统的可靠性主要考虑软件在操作使用中不发生失效的概率。当软件不能按照规格说明运行时，即发生失效。对软件可靠性进行预测需要：①软件预期操作使用的模型；②模拟实际操作条件的测试环境；③测试数据分析及可靠性推断的方法。

在统计测试中，首先需要建立能够表征软件使用情况的模型，然后利用模型在软件所有可能操作中进行统计上正确的采样。以样本运行结果为基础，得出软件操作可靠性结论。

事实上，通过人为选择测试用例进行测试，是很难得出软件可靠性结论的，而需要通过统计测试的方法科学地验证软件的可靠性。实验结果说明，随机测试优于输入域划分测试和结构覆盖测试，随机测试具有更好的效费比，并使测试人员能够对软件可靠性作出可靠的估算。但随机测试可能无法完全测试软件处理异常情况的能力。

软件使用模型刻画了软件系统的操作使用，这里的操作使用是指软件在预期环境中的预期使用情况，即从中抽取统计上正确的测试用例样本。使用模型以软件的规格说明为基础，因此在编写代码之前，建立使用模型所需的信息已经存在，软件开发和建模可以独立进行。制定测试计划可以和软件开发并行展开，这样可以缩短软件开发和发布的时间。使用模型通过使用概率提供了每项功能的重要性数据，为测试人员构造有效的测试用例提供了所需信息，通过统计测试为软件可靠性推断提供了依据。

国内外通过对大量软件产品失效数据的分析发现，绝大多数失效都是由相对很小一部分软件错误造成的，而且经常报告的失效都是在软件使用的早期出现的。因此，基于软件使用模型的统计测试可以保证在实际使用中最常发生的失效能够在测试早期被发现，以使建立和应用使用模型的效费比可达 10 或更高。因此在有限的测试预算下，基于使用模型的统计测试是最有效的测试方式。

10.2.2　操作剖面的构造

建立操作剖面是一个迭代的过程，对软件规格说明或预期使用的任何改变都必须进行评审，并对操作剖面进行适当的修改。下面是建立操作剖面（使用模型）的具体方法。

1. 了解规格说明

操作剖面（使用模型）反映了软件规格说明定义的预期使用情况。因此在建立使用模型之前，必须对规格说明进行评审和澄清，保证规格说明提供了完整的功能规格说明，描述了软件在预定环境中的预期使用情况。

2. 明确预期的用户、操作及环境

软件是在某个环境中由用户操作使用的，定义用户、操作及环境也就确定了软件测试认证的对象。

1）用户

用户是指与软件相互作用的任何实体，即施加激励和接受响应的任何实体。例如，用户可能是另一个软件、人员或硬件设备。

描述个人用户的参数包括：①工作描述，如操作人员等；②对各种系统资源的访问权限；③对使用环境的熟悉程度，即初次使用者、偶尔使用者、有经验用户；④有关使用领域的知识，即与软件应用领域有关的技术储备。

表征硬件用户的参数包括：①硬件平台的物理特性；②软件配置特性，如操作系统、通信

软件；③对硬件传输带宽的使用特性。

2）操作

软件的操作可以是一次击键、工作对话、事务，或任何其他适当的软件服务执行单位。可以根据操作的处理方式进一步定义操作，如对于某特定软件工具，不论是由单用户使用还是由多个用户同时使用，从用户角度来看都是一样的。因此为了保证软件对这两种操作都能正确处理，需要对软件进行单用户和多用户访问测试。操作的处理类型包括：①菜单驱动与文本查询；②交互处理与批处理；③信息系统与实时处理；④多用户与单用户。

3）环境

软件使用环境可以通过很多种因素来描述，其中包括：①与其他系统的交互（软件、数据库、I/O 设备等）；②计算机系统负载（内存、CPU、网络负载等）；③外来数据的完整性；④同时访问数据的要求。

3. 划分用户和环境参数

操作通过多个参数来定义，参数有不同取值，包括同时施加的多个激励，其组合数量非常大。定义的使用模型需要满足的要求包括使用模型要足以描述整个系统，使用模型要足够简单。使用模型只有足够简单，才能有效地对其进行验证，使统计测试具有较高的效费比，并对基于使用模型的软件可靠性进行认证。要达到这些相互冲突的目标，定义使用模型时需要对用户和环境参数进行划分。

将各种操作分类之后，建模人员需要确定如何在使用模型中对这些分类进行表示，保证测试环境与被测操作类假设相对应。对操作类建模至少有两种方式：①对于每类用户建立不同的使用模型；②建立单个使用模型，对每个用户类的总体分布作出判断，以不同的路径对应不同的用户类。

基于使用模型的统计测试完成后，根据测试结果作出的推断必须以事先的假设为根据，包括划分操作类的假设、基于分类的使用建模假设。例如，某在线软件工具的使用模型以用户的操作经验作为划分用户类的依据。初次使用的新手在使用某项功能时，100 次中有 80 次要访问在线帮助。但随着使用次数的增加，访问在线帮助的频率逐步减少，最终成为有经验的用户。平均来说，新手在使用此功能时，100 次中有 30 次要访问在线帮助。

如果对于每种操作，按照各用户类中一般用户平均使用情况定义转移概率，那么在新手类的使用模型中，应该包含一项从此状态到在线帮助的 30% 转移概率。根据此使用模型进行统计测试，其结果可以对软件工具在很多新手大量使用下的可靠性作出推断。但是，这些统计测试结果不能用来对软件工具在个别新手初次使用情况下的可靠性作出推断。

4. 确定所需的操作剖面粒度

随着使用模型粒度的提高，使用模型开发和维护的代价也趋于增加。为了确定合适的粒度级，建模人员应该对建立详细模型的代价及其对测试的改进效果进行权衡分析。

在开发过程中，规格说明的详细程度通常是不断提高的。采用较抽象的使用模型有利于在开发阶段早期进行建模工作，以后对模型的修改也更加方便，详细的使用模型对软件实际操作的描述更精确。假定使用模型是正确的，使用模型越详细，由统计测试结果推断的可靠性就越接近于软件实际操作的可靠性。

在有些情况下，整个使用模型的粒度可以不必一致。例如，建模人员可能对关键性软件或新增软件更详细地加以描述，而对于非关键性软件或成熟软件的描述较粗略。影响使用模型粒度的其他因素还有输入空间的划分情况等。测试认证要求可以体现在可靠性估算过程，操

作、用户和环境的分类，必须执行的测试用例数量，规定的预算和日程等。使用模型规模的有关度量可以包括状态数、弧线数等。根据经验，分析这些要求和度量与模型开发维护代价之间的关系有助于确定最佳的模型粒度。

5. 确定操作剖面结构

操作剖面结构为事件与事件间的转移模型，这里的事件可以是具体的激励（来自用户或环境的输入），也可以是抽象的占位符（表示软件使用的状态）。操作剖面结构，即使用模型结构，以软件的规格说明为基础，需要反复修改，逐步完善。除了用户直接输入的激励，模型结构还必须包括影响系统操作的外部激励（如来自外部系统或文件的输入）。随着软件开发的不断深入，和对软件使用操作的更多了解，使用模型的结构也随之不断完善。

使用模型结构可表示为图、形式文法或马尔可夫链。对于使用模型的三种表示方法，我们通过一个简单的交互式软件系统加以说明。这个系统的主窗口包括两个按钮“显示”和“退出”，单击“显示”按钮弹出对话框，单击“退出”按钮则退出系统。对话框中只有“确定”按钮，单击“确定”按钮返回主窗口。对于这个系统，使用模型可能包含 4 个使用状态：启动（Inv），主窗口（MM），对话框（Dsp），退出（Term）。

1）图

使用模型（如图 10-1 所示）可以表示成由节点和弧线组成的图，其中节点为使用状态，弧线表示状态之间的有序转移。图的表示方法易于用户理解，因而可用于使用模型的验证。

图 10-1 简单交互式系统的图结构

2）形式文法

形式文法可以准确地描述软件使用情况。形式文法由三部分组成：①终结符的有限集合；②非终结符的有限集合，非终结符由非终结符和终结符递归地描述；③关于终结符和非终结符的规则，称为生成式。

上述简单的交互式软件系统可以通过形式文法描述如下：

```
<Random usage> -> <Inv><Use><Term>
<Inv> -> "invoke main window"
<Use> -> <MM>{<Dsp><MM>}
<MM> -> "invoke display screen"|<Term>
<Dsp> -> "return to main window"
<Term> -> "exit from system"
```

其中，非终结符包含在＜＞中，终结符包含在""中，生成式表示为－＞，|表示布尔 OR，{...}表示括号内容的零次或多次出现。

3）马尔可夫链

马尔可夫链包含唯一的初始状态（代表软件启动）、唯一的终止状态（代表软件终止）、中间使用状态的集合、状态之间的转移弧线。马尔可夫特性要求在给定当前状态情况下，下个状态与所有过去状态无关。马尔可夫链的状态转移可以通过二维矩阵表示（如表 10-1 所示），其中状态标号作为索引，弧线概率作为表值。为了利用马尔可夫链不可分的特性，必须保证每个状态都有下个状态（终止状态的下个状态为启动），每个状态的离开弧线概率和为 1。转移矩阵为方阵，每行的和为 1。对于上述例子，Inv 为唯一的初始状态，Term 为唯一的终止状态，MM 和 Dsp 为中间使用状态。表 10-1 给出了该系统的转移矩阵。在转移矩阵中，方格（i，j）中的 X 代表从状态 i 到状态 j 的转移概率。

表 10-1 状态转移

	Inv	MM	Dsp	Term
Inv		X		
MM			X	X
Dsp		X		
Term	X			

6. 验证结构的正确性

在建模过程继续进行之前，必须对照软件规格说明对操作剖面结构加以验证。对于系统预期的使用环境、用户类、关键性功能，或其他可能影响操作剖面结构的参数，此时必须确定并进行验证。

7. 确定概率分布

建模的下一步是为模型结构中每个转移分配相应的概率，转移概率集合决定了软件输入域的概率分布，此概率分布是进行统计测试的基础。

转移概率是以软件预期使用为根据确定的。一般操作概率有三种方法：①根据现场数据分配转移概率；②根据软件预期使用假设分配转移概率；③如果没有软件预期使用信息，对于每个状态的离开弧线，可以分配相同的概率。

马尔可夫特性要求下个状态仅取决于当前状态。因此在软件的运行过程中，如果某个状态可能被多次调用，而且每种调用的概率又不相同，这时建模人员需要决定给每种调用分配平均概率还是分配不同概率，不同的选择会使模型结构有所不同。例如再考虑上述简单的例子，假设对于任意 10 次使用，其状态序列的分布如表 10-2 所示。

表 10-2 状态序列分布

状 态 序 列	次 数
Inv，MM，Dsp，MM，Term	8
Inv，MM，Dsp，MM，Dsp，MM，Term	1
Inv，MM，Term	1

如果在确定使用概率时不考虑输入是否相互独立，操作概率的分配如表 10-3 所示。

表 10-3 状态转移及概率(第一种模型)

从状态	到状态	频 率	概 率
Inv	Mm	10	1.0
MM	Dsp	10	0.5
MM	Term	10	0.5
Dsp	MM	10	1.0

此模型结构对应于图 10-1 给出的图。但是仔细检查表 10-2 中的数据，可以看出从 MM 到 Term 的转移概率取决于 Dsp 是否被事先调用。为了更准确地描述实际的使用情况，可以将 MM 分解成两个状态：MM0 表示对主窗口的初始调用，MM1 表示对主窗口的所有后续调用。新的模型结构对应于图 10-2 给出的图，表 10-4 给出了相应的使用概率分布。

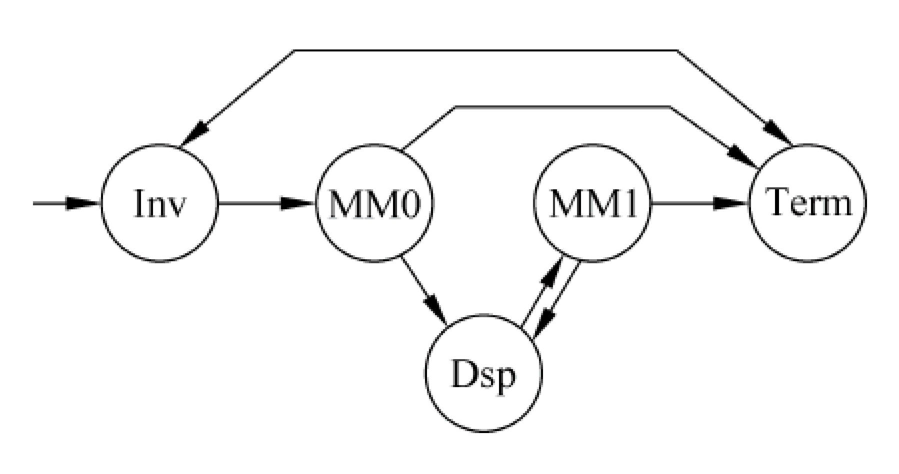


图 10-2 简单交互式系统的另一种图结构

表 10-4 状态转移及概率(第二种模型)

从状态	到状态	频 率	概 率
Inv	MM0	10	1.0
MM0	Dsp	9	0.9
MM0	Term	1	0.1
MM1	Dsp	1	0.1
MM1	Term	9	0.9
Dsp	MM1	9	1.0

尽管新的模型准确地描述了观察到的状态序列,但是应该注意的是,在观察到的状态序列中,只有一个序列包含对话框的多次调用,此模型假定这样的调用可以无限多次。模型还假定在首次调用对话框并返回主窗口后,所有从主窗口到对话框的转移概率是不变的。

如果要更准确地描述软件使用情况,某次调用对话框的概率应该小于此前各次调用对话框的概率。要在模型中把这个问题考虑进去,还需要进一步分割 MM 状态。但是模型越详细,建模和维护的代价就越高,因此需要折中考虑。

8. 验证概率分布的正确性

建模人员必须根据软件预期使用情况的已有信息对概率分布加以验证。随着软件的不断成熟和对软件预期使用情况的不断了解,建模人员应该对概率分布不断进行完善。

10.2.3 操作剖面的使用

基于使用概率分布的随机测试可以保证最经常使用的操作平均得到最多的测试。然而对于软件的某些特殊操作,不论其使用概率大小,也都要保证得到足够的测试,例如关键性的操作功能、新实现的操作功能等。

1. 关键性操作的建模

对于大型软件操作剖面中的关键性操作,有三种处理方式:①对于各种不常使用的关键性操作,分别建立使用模型,并以此为基础进行测试;②对于不常使用的关键性操作,进行覆盖测试或启发式测试(但是根据这种测试的结果无法对软件可靠性作出科学的推断);③调整使用概率分布,保证关键性操作得到更多的测试(此时的使用分布不是对预期使用情况的准确描述,将影响对软件操作可靠性的推断)。

2. 新增操作的建模

在成熟软件基础上增加新内容时,新增软件在现有操作剖面中成为新的组成部分。随着软件增量式开发的进行,软件系统的规模越来越大。此时对软件系统进行测试,结果是最新程序得到的测试越来越少,因为新程序在整个软件中所占的比重越来越小。下面三个方法可以在一定程度上解决这个问题:①为新增软件建立单独的使用模型;②调整模型粒度,使旧操作的描述更抽象,新增操作的描述更为详细;③调整使用概率分布,保证新增软件得到更经常的测试(此时的操作分布不是对预期操作的准确描述,这将影响对操作可靠性的推断)。

3. 操作剖面的使用过程

制定统计测试计划、进行软件统计测试认证的过程包括这些步骤:①进行效费分析;②建立预定系统环境;③确定测试的终止准则;④根据使用模型生成随机测试用例;⑤确定测试用例结果的评判方法;⑥进行软件统计测试,记录观察到的失效;⑦分析测试结果,在此基础上对软件操作可靠性作出推断。

在根据统计测试结果进行可靠性估算时,要考虑到相应的假设,包括有关软件规格说明的

假设,有关划分操作类型的假设,有关使用建模的假设,以及测试过程中的假设等。

10.2.4 基于操作剖面的软件可靠性疲劳测试

前面介绍的软件可靠性测试是基于被测软件操作剖面的统计测试方法,这种方法在操作剖面确定后如何针对由于长期使用软件性能下降,甚至完全失效这种严重影响软件可靠性的问题上有一定的不足。例如,无休止的线程、无释放的文件锁闭、数据污染、存储空间的彻底分裂与积聚差错等。而这些问题的产生归根结底还是软件设计和逻辑实现带来的问题。如果这些问题解决了,就不会出现由于长期使用使软件性能下降的问题。而解决这一问题的有效方法就是基于操作剖面的疲劳测试,即在一段时间内(经验上一般是连续72小时)保持操作剖面上的某些操作频繁使用,检查系统是否发生功能或者性能上的问题。

通常现代大型软件的操作剖面很复杂,覆盖整个操作剖面进行疲劳测试是不现实的。如何在被测软件的操作剖面上确定疲劳测试的范围或测试点,以及疲劳测试的类型是软件可靠性疲劳测试的重要研究内容。因此,在传统的操作剖面上结合软件可靠性统计测试方法,针对软件有别于硬件的自身特性,以及软件由于长期使用使软件性能下降,甚至完全失效这种严重影响软件可靠性的问题,采用操作剖面疲劳测试的方法,确定重要的疲劳测试点,并在这些疲劳测试点上分别进行相应类型的疲劳测试,以保证软件经过长时间运行后,性能不会下降,失效不会产生,来达到提高软件可靠性的目的。这种面点结合、互相补充的测试思想丰富了软件可靠性测试内容,解决了软件可靠性测试不仅要考虑操作剖面的功能测试问题,还要考虑操作剖面的疲劳测试问题,并使操作剖面和操作剖面上的疲劳测试点成为相辅相成的软件可靠性测试方案。

可以看到,准确地寻找操作剖面上的各个疲劳测试点,有效地确定疲劳测试类型,设计相应的疲劳测试用例,并将这些测试用例纳入到被测软件的操作剖面的统计测试用例中是软件可靠性疲劳测试的关键,也是现代软件可靠性测试的关键。这需要在操作剖面的基础上对被测软件进行静态分析和动态测试,查找可能严重影响软件可靠性的因素,如内存泄露、数据污染、线程死锁、文件冲突等,并将这些作为疲劳测试点的疲劳测试内容,确定相应的疲劳测试类型。

总之,由于软件自身的特点,软件可靠性与硬件可靠性相比内涵发生了根本的变化,软件可靠性估算必须以软件可靠性分析及软件可靠性测试等结果或缺陷数据为重要依据,软件可靠性估算模型及模型输入参数与软件可靠性测试结果紧密相连。为此,我们一定要抓住软件的特性,强调软件可靠性测试对于软件可靠性估算的重要支撑作用,研究软件可靠性测试方法和技术,特别是研究基于被测软件操作剖面的疲劳测试技术和基于软件可靠性分析和测试结果的软件可靠性估算模型,这是一个有理论价值和变革意义,并存在很大的难度且面临极大挑战的研究课题。

10.2.5 结论

基于使用模型的软件统计测试方法可以使有限的测试资源得到最有效的利用,这种测试方法也是唯一能直接对软件操作可靠性进行估算的统计方法。通过操作建模可以改进规格说明,得到规格说明的分析性描述,量化测试代价,通过统计测试为软件可靠性推断提供依据。

目前,基于使用模型的统计测试方法成功应用在多家公司和政府部门的项目中,在操作剖面的建立、分析、使用,以及统计测试等方面的经验正在不断积累,应用环境涵盖了实时嵌入式代码及复杂的软件工程环境。

Whittaker 和 Poore 关于马尔可夫模型的研究表明，利用马尔可夫模型生成随机测试用例可以获得有关测试过程的大量有用信息。这项工作为研究更有效的模型设计、测试用例生成和测试工作量分配方法提供了基础。研究人员仍然在操作建模、测试方法、可靠性估算等方面继续努力，以期发现新的方法，大幅度降低测试代价，并提高基于模型的测试验证成效。

10.3 软件可靠性测试工具

目前，软件可靠性评估或测试工具主要有 SMERFS、AT&T SRE Toolkit、SREPT 以及 SRMET 等，除 SREPT 外，这些工具基本上将被测软件视为黑箱。

SRMET 是由航天软件评测中心研制开发的，用来评估软件可靠性及维修性的工具，其着眼于软件故障的发生时间和发生次数，通过计时模型和计数模型的分析，把测试过程中的软件可靠性数据进行整合处理后，可有效地进行软件的可靠性和维修性评估。

SRMET 工具进行可靠性、维修性评估是基于以下假设。

(1) 软件的每一次故障都是由程序中的一个错误引起的。

(2) 故障发生时间是相互独立的。

(3) 故障一旦发生，则立即投入查错，并及时得到改正。

(4) 改错不会引入新的错误。同时，对于试验数据有以下要求：只考虑故障发生时间的数据形式，这里把被测软件的故障暴露时间序列作为有效的试验数据，用变量 t_i 表示第 i 个故障发生的时间($i=1,2,\cdots,n$)。

SRMET 具有图形、表格、全中文显示等良好的用户界面，所有评估模型的解算都依据严格的理论分析给出了优化算法，计算效率最高、精度最好，同时对软件可靠性模型具有最适合模型的推荐功能。

SRMET 根据用户提供的软件故障测试数据，通过对故障数据调整和分析、软件可靠性模型选定、模型参数估计等工作，评估及预测待评测软件的可靠性。软件可靠性评估和预测工具提供了故障数据收集工具，并将评估和预测结果以图形方式直观地显示给用户。

SRMET 在用户提供原始故障数据、操作剖面、测试数据分类信息的基础上围绕“可靠性测试失效数据生成”和“可靠性模型选择专家系统”展开，具体内容包括故障数据分类、操作剖面确定、测试计划调整建议、失效数据的拟可靠性测试变换、可靠性模型选择专家系统、组合模型。

原始故障数据根据实际应用需求按故障类型、故障数统计和故障发生时间进行分类并录入到数据库中。

操作剖面是由操作及其发生概率所构成的集合，它是以表格的方式进行收集，每一项包括操作名称及该操作的发生概率。

测试数据分类信息将用户在测试计划中设计的测试用例和操作剖面中的操作进行对应，其测试数据分类信息以表格的形式输入，分为两栏：一栏是每个操作的名称，一栏是测试相应操作所使用的用例数目。用户可以以交互的方式逐步输入测试各操作所使用的用例数目，用户也可使用其他工具统计好以后，直接输入测试各操作所使用的用例数目。

由于传统的测试与软件可靠性测试所要求的基于操作剖面随机测试方法有一定距离，为了向软件可靠性测试的要求看齐，可采取逐步过渡的方法。即在保证传统测试完整实施的基础上，要求系统可以根据确定的操作剖面对测试计划提出适当的调整建议，在可能的情况下测试计划将进行相应的调整。如给出测试计划中的测试用例与操作的对应关系，根据操作剖面

和测试计划中各个操作所对应的测试用例数目给出测试计划的调整建议。其基本思想是：测试用例的总数将以操作剖面中表明的比例实施分配，并在调整建议中还应考虑小概率操作的用例情况。

通过非软件可靠性测试收集到的故障原始数据必须经过分析和调整，以生成软件可靠性模型适用的故障数据。而利用被测软件的操作剖面等信息，依据合适的统计方法，由非软件可靠性测试所得到的故障数据生成可用于软件可靠性评估和预测的拟软件可靠性测试故障数据。拟可靠性测试变换的输入是操作剖面、变换前故障数据、测试用例分配情况，输出是变换后的拟可靠性测试故障数据。

根据故障数统计类数据和故障发生时间类数据对应的计算模型进行分别或组合计算，可实现对软件可靠性的定量与定性预测。目前 SRMET 支持的计算模型有 Schneidewind 模型、非齐次泊松过程(NHPP)模型、S-型可靠性增长模型、Brooks_Motley 模型、广义泊松模型、Shooman 模型、Jelinski_Moranda 模型、几何衰减过程模型、Musa 基本模型、Musa 对数泊松模型、Littlewood_Verrall 线性模型、Littlewood_Verrall 多项式模型。

习题

1. 什么是软件可靠性测试，一般采用何种方法进行软件可靠性测试?
2. 什么是操作剖面，什么是统计测试，如何基于操作剖面进行统计测试?
3. 怎样构造操作剖面，如何运用操作剖面进行软件可靠性测试?



第11章 软件安全性/软件安全测试

我们今天的现代化社会十分依赖于软件，软件可以实现企业的商务流程，可以操控我们日常中所有的电子商务（如手机、飞机和取款机），软件也是一些重要服务的核心，例如电话网络、互联网、空中交通指挥系统等。对全球经济增长、人民生活质量、企业成功以及政府组织机构来说，如今软件比以往任何时候都重要，我们必须使用安全的软件。但为了能够得到真正安全的软件，软件的安全测试是非常重要或必不可少的环节，因为，软件安全测试能够大幅提高应用软件的安全。

当前软件安全包含两重含义：Software Safety 和 Software Security。

Software Safety（软件安全性，或称软件安全保险）是指软件在系统中运行而不至于在系统工作中造成不可接受的风险的能力，如人身伤亡、设备损坏、财产重大损失、严重污染环境等。

Software Security（软件安全，或称软件安全保密）就是使软件在受到恶意攻击的情形下依然能够继续正确运行及确保软件在被授权范围内能够合法使用。

11.1 软件安全性测试

软件本身不会造成危险，但当软件用于过程监控、实时控制、武器、航天、医疗、核反应等方面时，软件的错误能够通过硬件与软件接口使硬件发生故障，从而造成严重事故。这类软件称为“安全性关键软件”。为解决这类问题而采用的一系列方法技术称为软件安全性技术。

软件没有硬件所具有的物理和化学属性，因此它对人类和社会没有直接威胁，不会造成直接的损害。但是当软件用于过程监测和实时控制时，如果软件中存在错误，则这些错误有可能通过硬件与软件的接口使硬件发生误动或失效，造成严重的安全事故。

由于许多隐蔽性强的或非多发性的错误很难被设计人员和测试人员察觉，仅仅依靠设计技术的改进仍然不足以解决安全性问题，这就需要一套严格的安全性分析程序和安全性分析方法，以预防安全事故发生或在发生事故时减少危害程度，软件安全性工作的意义和价值正在于此。

11.1.1 软件安全性概念

软件安全性是安全性关键（Safety-Critical）软件密集型系统（如综合航电系统）的一个重要质量要素。

安全性关键系统（Safety-Critical System，SCS）泛指具有潜在破坏力的一类系统。此类系统一旦失效，就可能造成严重的后果，如人员伤亡、财产损失或环境破坏等。近年来，软件在SCS中的应用越来越广泛。从航空航天领域到核工业领域，从电力系统到医疗系统，软件承担

着安全性关键的指挥控制功能。同时软件在 SCS 中的应用规模也与日俱增。例如，在 F-22 战机的综合航电系统，软件实现的航电功能高达 80%，软件代码达到 170 万余行。而在 F-35 战机的先进综合航电系统中，软件代码达到 500 万～800 万行。这表明，越来越多的 SCS 日益软件密集化，逐渐形成安全性关键的软件密集型系统。

另外，由 SCS 软件引发的事故或事件却频发不断：发生在 20 世纪 90 年代的 Ariane 5 运载火箭、SOHO 太空船等 5 起航天器事故的罪魁祸首是软件；2004 年 12 月 20 日，一架 F-22 因飞行控制软件故障而坠毁；2007 年 2 月 11 日，12 架 F-22 在穿越国际日期变更线时也是因软件缺陷问题造成导航故障，战机被迫在无导航和通信能力下危险返航；在民用上，美国放射治疗机软件错误致五名患者受超剂量辐射死亡；美国血液数据库程序出错，致使被 AIDS 污染的血液被用于治疗。由此，软件安全性已成为决定 SCS 安全与否的重要因素之一。

1. 软件安全性定义

软件安全性工作的出发点是系统安全性。一个单独的软件本身并不存在安全性问题。只有当软件与硬件相互作用可能导致人员的生命危险、或系统崩溃、或造成不可接受的资源损失时，才涉及安全性问题。

软件安全性定义有多种表述，如：①美国宇航局的软件安全性标准中定义软件安全性是"在整个软件生命周期运用系统安全性工程技术来确保软件采用提高系统安全性的有效措施，并确保那些可能降低系统安全性的错误均已被排除或控制在可接受的风险水平"；②欧空局(ESA)组织的一项研究报告中定义软件安全性是"软件在系统中执行而不致在系统工作中造成不可接受的风险的能力"；③GJB/Z142—2004《军用软件安全性分析指南》中定义软件安全性为"软件具有的不导致事故发生的能力。确切地说，软件安全性是软件的功能安全性"。

上述定义尽管表述不同，但有两点是共同的。

(1) 强调要在系统环境中讨论软件安全性。

(2) 软件安全性是软件的一个质量属性或一种能力。软件安全性离不开系统安全性，特别是在 SCS 软件系统中更是如此。软件安全性需求来源于系统安全性要求，只有通过对复杂系统逐层的危险分析，才能确定系统所面临的危险，只有依据对具体系统设计方案的分析，才能确定软件与相关危险的关联度，在此基础上才能进一步分析确定软件的安全性需求和软件的关键等级。此外，软件安全性是软件的一种质量属性，是通过软件安全性的设计和分析而形成的。当软件安全性需求确定后，需要在软件开发的各阶段采用相应的软件安全性开发和分析方法来实现和验证软件安全性需求。

2. 软件安全性工作

软件安全性工作要贯穿软件生命周期的始终，在软件开发的各个阶段都需要开展相应的软件安全性工作。

美国军标 MIL-STD-882B 将软件系统的安全性工作归结为以下几项：

(1) 确定系统及系统中软件的安全性要求。

(2) 将系统安全性说明中的要求准确转化为系统或分系统说明的要求，转化为软件需求说明的要求，并将这些要求在软件设计和编码中实现。

(3) 在系统、分系统说明及软件需求说明中确定当可能发生安全事故时的系统决策，这些决策包括失效安全、失效降级使用、失效容错使用等。

(4) 确定软件系统中的安全关键单元，安全关键单元是指那些对系统安全性有关键性影响的程序、分程序和模块。

(5) 对软件的安全关键单元进行分析。

(6) 通过分析、验证、确保软件系统安全性要求的实现,验证不存在有损于安全性的单个或多个失效事件,保证系统的安全性要求不致引起新的危险。

(7) 确保编制的程序不会因为触发危险功能,或阻碍正常的功能的执行而使系统处于危险状态。

(8) 保证对系统进行充分的安全性测试,包括失效事件发生的测试。

可以看出,软件安全性工作可以分为软件安全性开发、软件安全性分析两类。

软件安全性开发工作的主要内容是确定软件安全性需求、开展软件安全性设计、编程和测试,其目的是要全面、正确、合理地确定软件安全性需求,并通过采用一系列的软件安全性设计和编程方法与准则,在软件设计和实现阶段将软件安全性需求设计进去并加以实现。

目前,确定软件一般安全性需求清单、制定软件安全性设计准则、定义软件编程语言的安全子集或编程准则等都是工程上常用的方法。

软件安全性分析工作的主要内容是通过采用一系列的软件安全性分析方法,来验证软件安全性需求的正确性、合理性和完备性,验证软件安全性设计和编程的正确性、软件安全性测试的充分性,其目的是要验证软件安全性需求是否通过软件安全性的设计、编程和测试被全部正确实现,另外通过软件安全性分析工作能发现软件安全性开发工作的不足,进一步完善和改进软件安全性开发工作。

目前,工程上常用的软件安全性分析方法如下:软件安全性流向分析,时间、吞吐量和空间分析,独立性分析,设计逻辑分析,设计数据分析,设计接口分析,测试覆盖分析,测试结果分析,软件故障树分析(SFTA),软件失效模式及影响分析(SFMEA)等。

11.1.2 软件安全性分析

如果我们称可能导致不可接受的风险发生的异常条件为危险条件的话,那么按照前面安全性定义,我们可以认为安全性是指系统在规定的条件下、规定的时间内,完成规定功能的过程中避免危险条件发生的能力。

从软件开发特别是软件设计角度看,保证安全性的关键在于系统设计时综合考虑全部外部因素(包括各种异常条件),合理地设计、定义和分配各子系统的功能、时序及其相互之间的接口,使得危险条件出现时系统可以处于一种安全的状态,并将可能的损失降低到最小。

由于操作人员的错误、硬件故障、接口问题、软件错误或系统设计缺陷等很多原因都可能影响系统整体功能的执行,导致系统进入危险的状态,故系统安全性工作自顶向下涉及系统的各个层次和各个环节,而软件安全性分析是系统安全性工作中的关键环节之一。

1. 软件安全性分析的内容

从工程角度看,安全性分析的实质是通过对安全性关键软件的运行环境、设计结构和测试结果等进行全方位的分析,发现软件中与系统危险条件相关的设计缺陷及危险产生条件,获得与软件相关的系统危险模式,并分析危险的严重等级与发生概率,最终确认系统的危险风险指数,给出安全性评价。因此,软件安全性分析不能只从软件本身出发,必须从系统角度进行分析,考虑软件使用过程中软件、硬件和操作人员的相互作用,分析软件可能的工作时序、适用条件、逻辑缺陷及其可能造成的不利影响。

分析的对象:软件的需求和设计文档、软件测试结果,以及与软件相关的接口、人员操作、硬件状态、硬件故障、系统时序等。

MIL-STD-882B 规定软件的安全性分析工作包括软件需求危险分析、概要设计危险分析、详细设计危险分析、软件编程危险分析、软件安全性测试、软件与用户接口危险分析及软件变

更危险分析。

1）软件需求危险分析

软件需求危险分析是利用系统初步危险分析的结果，初步确定软件的安全关键单元。要点为：

（1）建立软件安全性需求的跟踪系统，记录每个需求的实现情况。

（2）从安全性的角度评审系统说明和分系统说明，评审软件需求说明、接口说明，以及其他有关系统方案和要求等文件。

（3）将系统安全性的要求分配到软件。

（4）由系统的初步危险表导出软件的危险表。

（5）分析功能流程图、编程语言、数据流图、存储和时序分配图表以及其他的程序文档。

2）概要设计危险分析

概要设计危险分析的主要分析工作有：

（1）从软件的危险表出发，分析其中的危险事件与软件的组成单元之间的关系，并将这些危险事件有关的软件单元确定为软件安全关键单元。

（2）对软件进行检查，确定软件的各个单元、模块、表、变量之间是否相关，确定相关的程度（凡是对软件安全性单元有直接和间接影响的其他单元，也要确定为软件安全关键单元，并且分析它们对安全的影响）。

（3）分析软件安全关键单元的概要设计是否符合安全性的要求，分析的结果应送交给软件设计人员和项目主管。

3）详细设计危险分析

详细设计危险分析是安排在概要设计初步评审之后、软件编码之前，分析的结果交给关键设计评审：

（1）依据需求危险分析、概要设计危险分析确定的危险事件，分析这些事件与低层次的软件单元的关系，将对危险事件有影响的单元确定为软件安全关键单元，分析这些单元对危险事件影响的方式和途径。

（2）在低结构层次上考察软件的各个单元、模块、表和变量之间的相关程度，将直接和间接影响软件安全关键单元的其他单元确定为安全关键单元，分析它们对安全的影响。

（3）分析软件安全关键单元的详细设计是否符合安全性设计的要求，分析的结果应该送给软件设计人员和项目主管。

（4）确定在测试计划、说明和规程中需要包含的安全性要求。

（5）确定在系统操作员手册、软件用户手册、系统诊断手册及其他手册中需要包含的安全性要求。

（6）确保编程人员了解安全关键单元，向编程人员提供有关安全性的编程建议和要求。

4）软件编程危险分析

软件编程危险分析考察软件的安全关键单元以及其他单元的源程序和目标程序是否实现了安全性设计的要求，该工作与编程同时进行。考察的内容有：

（1）考察软件安全关键单元的正确性，考察它们对输入或输出时序、多重事件、错误事件、失序事件、恶劣事件、死锁及输入数据错误的反应和敏感性。

（2）考察程序、模块或单元中是否存在影响安全性的编程错误。

（3）考察安全关键单元是否符合系统说明、分系统说明和软件需求说明中提出的安全性对策，这种考察必须在源程序和目标程序中进行。

(4) 考察软件安全关键单元的安全性设计要求的实现情况,确保达到所要求的目标,确保硬件和其他模块的失效不致影响软件的安全性特征。

(5) 使系统在危险状态下运行,并考察硬件或软件失效、单个或多重事件、失序事件、程序的非正常转移对安全性的影响。

(6) 考察超界、过载输入对安全性的影响。

(7) 评审正在制定的软件文档,确保这些文档包含了软件的安全性要求。

5) 软件安全性测试

软件安全性测试主要工作有:

(1) 对安全关键单元进行安全性测试,保证使危险事件发生的可能性降低到可以接受的水平。

(2) 向测试人员提供软件安全关键单元的安全性测试案例。

(3) 确保所有的软件安全关键单元按预定的测试方案进行安全性测试,准确地记录测试的结果。

(4) 除了在正常状态下进行的测试外,还要在异常的环境和异常的输入状态下测试软件,确保软件在这些状态下仍能安全运行。

(5) 进行软件强度测试,确保软件安全运行。

(6) 确保外购软件安全运行。

(7) 订购方所提供的软件,不管是否进行了修改,都需要进行测试,以保证这些软件在系统中安全运行。

(8) 确保在系统综合测试和系统验收测试中所发现的危险事件得到了纠正,确保对这些事件进行了重新测试,没有遗留问题。

6) 软件与用户接口危险分析

软件与用户接口危险分析主要包括:

(1) 提供检测危险征兆或潜在危险状态的方法,预防安全事故的发生。

(2) 控制危险事件,使其只有在特殊的情况下和操作员特定的命令下才可能发生。

(3) 向操作员、用户和其他人员提供报警功能,指示可能出现或正在出现的潜在危险。

(4) 当危险事件发生后,确保系统能够生存。

(5) 当预防和控制危险的规程失败后,或危险事件发生时,能提供控制损害程度的规程和恢复到安全状态的规程。

(6) 提供在严重危险状态下使系统生存和恢复功能的规程。

(7) 具有安全终止某个事件和安全终止程序运行的能力。

(8) 具有向操作员提供系统或软件失效的报警功能。

(9) 具有向操作员提供安全性决策所需的信息,确保危险数据能够明确显示。

7) 软件变更危险分析

软件变更危险分析主要工作有:

(1) 分析系统、分系统接口、逻辑和其他设计变更对安全性的影响,确保这些变更不会产生新的危险,不会触发已经消除的危险,不会使现存的危险变得更严重,不会对有关的设计和程序产生任何有害的危害。

(2) 对变更进行测试,确保更改后的软件不包含危险事件。

(3) 确保软件的变更已经在编程中准确地实现;评审和修改有关文档,以反映这些变更。

(4) 将执行软件变更危险分析的方法和程序纳入软件配置管理及变更管理计划。

2. 软件安全性分析的特点

软件安全性分析与系统安全性分析相比有以下特点。

(1) 软件安全性分析是系统安全性分析的一部分。软件安全性分析必须在系统安全性分析的基础上进行。

(2) 软件与硬件的故障模式不同,因此分析的重点也不同。硬件为物理产品,使用时有磨损。因此硬件安全性分析的重点为硬件故障以及共因失效所引起的系统危险状态。软件属于逻辑产品,无磨损。很多情况下并不是软件失效,而是在软件正常工作时,在某种特殊条件下软硬件相互作用导致的不安全情况。分析重点为软件设计缺陷,以及软件使用过程中软件、硬件和操作人员的相互作用。

(3) 与软件测试的侧重点不同。软件测试通常只能对一般的输入组合所涉及的执行路径进行验证,很少测试那些与某些特殊的输入组合和时序相关的软件故障。这种与软件的设计细节和软硬件交互时序等密切相关的软件故障必须采用软件安全性分析的方法。所以实际工程中,对那些可靠性要求较高难以进行可靠性验证的软件要进行安全性分析。

(4) 由于软件的逻辑、数据、时序等设计缺陷,与软件相关的硬件故障和状态等都有可能引起软件失效,导致系统进入危险状态。

分析时必须对软件进行全方位的分析,从系统顶层到软件的源代码,从外部的运行环境到软件内部的设计细节,包括软件的静态、动态、逻辑和物理模型。

(5) 软件安全性分析必须从系统角度分析软件可能的运行时序、运行状态、适用条件、逻辑缺陷及其可能造成的不利影响,这就不可避免地会涉及各种不同领域的专业知识与经验积累。

分析时以人为主,任何软件分析工具只能起辅助作用。这就对软件安全性分析人员提出了较高的要求。分析时要求有具备专门知识的软件安全性分析人员、熟悉系统结构的系统总体设计人员、软件设计人员、相关领域专家参加,共同工作。

(6) 由于软件的特点和运行背景的多样化,软件安全性分析没有固定的分析模式和分析程序,必须具体问题具体分析。

必须综合考虑软件的运行环境和总体要求,以及各种软件安全性分析技术的使用条件、适用范围和工作量,进行权衡分析,确定详细的软件安全性分析方案,方案包括分析的内容、方法、步骤和必要的测试验证环境和辅助工具等。

3. 软件安全性分析注意事项

软件安全性分析时应注意以下事项:

(1) 每一个软件的功能、规模、实现逻辑、运行环境等均具有其特殊性,每一种分析技术也有其适用条件和局限性。只有针对软件的特点和设计要求,综合考虑分析要素及工程限制,选择有效的技术手段,制定合理的分析方案,才可能获得满意的效果。

(2) 软件安全性分析要求不同领域的专家合作,从系统的角度考察软件、系统和操作人员之间的相互作用,因此人员配备时不仅要配备具备相关背景知识的软件安全性分析员,还必须配备对系统熟悉,且具有专门领域知识的系统总体设计人员。

(3) 软件安全性分析是从系统顶级开始,自顶向下逐层、逐级分析到软件功能层次,获得软件的安全性需求,再从软件的设计角度分析,自顶向下逐层分析软件设计缺陷对系统的不利影响。所以系统级、需求级和代码级的分析是不能缺少的,如果软件规模较大,软件结构设计分析也是必需的,甚至对系统、需求和设计的分析也可能转变为多层次的安全性分析。

(4) 分析内容的确定取决于软件本身的功能和系统设计要求,分析员根据分析内容选择

合适的分析技术，包括软件需求分析技术、软件可靠性分析技术和专门的安全性分析技术，对于比较复杂的系统甚至需要建立专门的仿真环境和分析工具进行计算机辅助分析。

4. 软件安全性分析常用技术

软件安全性分析常用的技术有功能危险评估（FHA）、初步危险分析（PHA）、软件失效模式及影响分析（SFMEA）及软件故障树分析（SFTA）这4种技术。

（1）功能危险评估是自上而下地确定系统功能故障状态，并对其影响进行评估的一种系统安全性分析技术。它系统综合地检查产品的各种功能，识别各种功能故障状态（包括功能丧失和功能故障），并根据其严重程度进行分类。

（2）初步危险分析是识别系统危险的一种技术，是进行安全性分析的一种重要手段。针对在程序设计、开发过程中需要跟踪、解决的危险及相关风险，PHA提供了危险清单的初步框架，并记录了通用的危险。

（3）软件失效模式及影响分析是对传统的系统分析技术失效模式和效果分析FMEA的扩展，是一种自底向上的分析技术，它以失效模式为基础，以失效影响或后果为中心，根据分析层次和因果关系推理、归纳进行分析，以识别软件的薄弱环节，并提出改进措施建议。

（4）软件故障树分析来源于传统的系统分析技术，故障树分析FTA是一种重要的软件安全性与可靠性分析技术，它特别适合在软件需求阶段进行分析。它是一种自顶向下的分析技术，选取显著影响系统的故障事件作为顶事件，对引起系统故障的各种软件原因进行分析。

11.1.3 软件安全性测试方法与技术

软件安全性测试是检验软件中已存在的软件安全性措施是否有效的测试，是保证系统安全性的重要手段。然而在工程项目中，常规的软件工程方法和软件测评手段并不能完全验证软件安全性，因此，需要通过软件安全性测试来验证与软件相关的系统危险已被消除或被控制在可接受的风险水平，并通过测试在软件中发现和排除隐蔽的重大错误。

1. 软件安全性测试内容

软件安全性测试应包含的内容有：①验证每一个软件安全性需求都有相应的软件安全性测试相对应；②证明每个软件安全性需求都通过一个或多个测试得到了满足；③通过测试分析软件实现对相关的风险进行了防范；④判断给出的软件安全性测试已足够充分。

2. 软件安全性测试流程

基于上述的安全性测试内容，软件安全性测试的主要流程如下：

（1）确认软件安全性测试规程。此步骤要求软件安全性工程师检查所建立的规程，并验证这些规程能否完全地测试软件安全性需求。

（2）执行和监督软件安全性测试。软件系统安全性团队必须监督软件安全性测试。既是为了确认软件安全性测试规程，也是为了判断该规程对软件安全性可能产生的异常影响。

（3）整理测试数据。实际上，在测试组将测试数据整理为有用的形式之前，大多数异常数据是很难发现的。数据整理的过程是通过抽取测试期间记录的数据来推导性能和其他参数，并表示为易理解的显示信息来实现的。

（4）分析测试数据。数据抽取和分析的目的是标识安全性相关的异常和产生不安全的因素。这些因素可能是设计、实现、代码、测试用例、规程和测试环境中的错误。

（5）重新测试失败的系统需求。通常情况下，通过一次测试并不能确认全部的软件安全性需求。在测试中仿真器、激励器或实验室环境能力的局限均可能使某些测试不通过。因此，需要通过回归测试来对该功能的安全性提供充分的保证。

（6）编写软件安全性测试报告。软件安全性测试报告必须标识所进行的测试，并分析测试的结果。在测试结论中，软件安全性团队使用该结果来更新安全性需求标准分析中的需求可追踪性矩阵，以及对系统和软件进行的各种初步分析和详细分析。软件安全性测试报告将被附加到系统安全性测试报告中。

软件安全性测试是一个独立的测试过程，其测试方法与一般测试相比具有自身的特点。

3. 软件安全性测试方法

目前国内外常用的软件安全性测试方法有3类：基于可靠性分析方法的、基于形式化模型的和基于软件测试方法的软件安全性测试方法。

1）基于可靠性分析法的软件安全性测试方法

基于可靠性分析法的软件安全性测试方法常用到的是基于FTA的软件安全性测试方法和基于Petri网的软件安全性测试方法。

（1）基于FTA的软件安全性测试方法是利用故障树的最小割集来生成软件安全性测试用例的方法。它以系统中最不希望发生的故障状态作为故障分析的顶事件，寻找导致这一故障发生的全部可能因素，绘制故障树。然后搜索出最小割集，并以最小割集为依据生成软件安全性测试用例。此外，由于应用的领域的不同，故障树中底事件的语义解释并不唯一，因此出现了一种基于形式化故障树分析建模的软件安全性测试方法。它将故障树的叶节点语义形式化为一个以时间为变量的实时间隔逻辑持续时间计算公式，消除故障树的语义模糊性，达到形式化故障树的目的。这种方法首先对软件需求规格说明进行分级划分，然后利用形式化故障树表示安全性需求。在搜索出形式化故障树所有最小割集的基础上，运用基于割集的安全性测试用例动态扩展算法进行用例设计。

（2）基于Petri网的软件安全性测试方法是利用Petri网简洁、直观、潜在模拟能力强等特点，在因果关系作用下进行推演的过程，体现系统的动态行为特征。目前，软件安全性测试存在两种基于Petri网的方法，即正向分析法和逆向分析法。正向分析法首先建立完整的可达图和状态标识表，得到Petri网的可达集，建立相应的Petri网模型，然后在可达集中搜索所有包含任意一个状态的状态标识，并将其标记为危险标识，从初始状态到该危险标识的每个变迁序列均可设计为一个测试用例。在生成用例时，对每个被标记的危险标识应至少生成一个用例。这些用例构成了针对该Petri网模型的软件安全性测试用例集。然而，正向分析的方法存在一定的缺陷，因为它需要生成完整的可达图和状态标识集，这对于逻辑和结构比较复杂的系统是比较困难的，甚至容易形成组合爆炸的问题，因此出现了逆向分析方法。逆向分析方法是构造所有可能导致危险的软件危险状态，然后分析求出由初始状态到该危险状态可能的路径，并通过构造测试用例来验证该路径是可行的。这说明逆向分析法要针对具体问题具体分析。

2）基于形式化模型的软件安全性测试方法

形式化方法的基本思想是建立软件的数学模型，并在形式规格说明语言的支持下提供软件的形式规格说明。目前，形式化软件安全性测试方法可分为两类：模型检验方法和定理证明方法。

（1）模型检验方法用状态迁移系统S描述软件的行为，用逻辑公式F表示软件执行必须满足的性质，通过自动搜索S中不满足公式F的状态来发现软件中的漏洞。模型检验技术基本思想是将系统表示成为自动机模型，并将系统要验证的属性用某种逻辑公式来表示，再采用穷举状态空间的方式来证明系统的模型是否满足要验证的属性。这种方法利用内置的结构形式分析（模型检验中的状态机描述）来驱动测试过程，给测试工程师提供了生成和评估安全性测试集的方法。

（2）基于定理证明的软件安全性测试方法将程序转换为逻辑公式，然后使用公理和规则证明程序是一个合法的定理。目前，由于定理证明过程非常耗时费力，所以一般只用于验证设计阶段的程序规范而非实际代码。

3）基于软件测试方法的软件安全性测试方法

基于软件测试方法的软件安全性测试方法常用到的是基于猜错法的软件安全性测试方法和基于接口语法的软件安全性测试方法这两种方法。

（1）基于猜错法的软件安全性测试方法依据经验、直觉和对被测试系统的探索兴趣，在按规则生成的测试用例集之外添加一些"另类"的用例。运用这种方法，为用例所涉及的实体定义一系列关系，借助这些关系实现用例的自描述，并最终和新的软件测试背景匹配，实现用例的再生。

（2）基于接口语法的软件安全性测试方法，根据被测软件功能接口的语法生成测试输入，检测被测软件对各类输入的响应。软件的接口包括多种类型：数据总线、文件、环境变量、套接字等，它明确或隐含规定了输入的语法。基于这样一种思想，接口语法测试定义了软件所接受的输入数据类型、格式。接口语法测试法的步骤是：首先识别被测软件接口的语言，定义语言的语法。然后根据语法生成测试用例。其中，生成的测试输入应当包含各类语法错误、符合语法的正确输入、不符合语法的畸形输入等。最后，通过执行测试检验被测软件对各类输入的处理情况，确定被测软件是否存在安全缺陷。接口语法测试法适用于被测软件有较明确的接口语法，易于表达语法并生成测试输入的情况。语法测试结合故障注入技术可以得到更好的测试效果。

4. 软件安全性测试技术——顶事件驱动的故障树分析法应用举例

处于故障树顶层的、具有严重后果的软件故障，称为顶事件；所有顶事件的集合，称为顶事件表；顶事件表中的每一个顶事件将有一个故障树与之对应。

1）软件规模较小的故障树分析法

根据程序中各语句的逻辑关系，分析顶事件的发生主要可能是哪些语句或模块造成的，并进一步往下进行分析直到某条语句或某个条件时为止，然后根据所有的逻辑关系画出它的故障树。

根据故障树确定顶事件的发生是由哪条语句发生错误或哪些条件组合而引起的，我们就可以根据这些分析的结果着重设计软件可靠性稳定增长测试和软件安全性测试的测试计划，并进一步选取合适的测试用例集合。

2）大型软件的安全性分析法

采取分层构造软件故障树的方法，也就是对程序的各个子功能模块进行分析，这样不仅建树和分析比较容易，而且可以找出软件的关键功能模块，以便对其进行重点分析。

大型软件由很多个功能模块组成，软件要执行一定的任务，根据用户的不同操作，软件将调用不同的模块。因而，应以功能模块作为底事件进行故障树分析。

采取"分而治之"的办法，在不同的层次上进行分析和建立故障树。"底事件"是针对整个软件系统的模块结构而言的，处于最下层的模块均为底事件。

在以功能模块作为底事件进行分析时，同样要确定一个系统的顶事件表。最后得出的故障树也很大，不能直接看出哪个是关键的功能模块。此时可采用最小割集法（一个最小割集代表系统的一种故障模式，即只要最小割集的事件发生，就会导致顶事件的发生），求出故障树的最小割集，从而在最小割集中找出系统的关键功能模块。

利用前面所提到的代码分析或其他软件分析技术对关键功能模块进行分析，找出可能导

致关键功能模块失效的原因。

根据故障重新计算顶事件的发生概率，达不到预定要求，则重新分析，根据变化了的情况重建故障树(由于程序改动以及某些关联的因素发生变化，因而第二次建的树可能和第一次不一样)，对它进行分析，找出另一个可能引起顶事件发生的关键功能模块。

11.2 软件安全测试

尽管在实际中软件故障通常是在没有蓄意攻击的情况下自然地产生的，但软件在遭受恶意攻击时能否正确地运行，现在越来越受到人们的关注。软件安全性(Software Safety)和软件安全保密或安全(Software Security)的区别在于是否存在专业人士恶意侵害、攻击和破坏系统。

软件安全属于软件领域里一个重要的子领域。在以前的单机时代，安全问题主要是操作系统容易传染病毒，因此单机软件安全问题并不突出。但是自从互联网普及后，软件安全问题愈加凸显，使得人们对软件安全测试的重视程度上升到一个前所未有的高度。

软件安全一般分为两个层次，即应用级别或应用软件的安全和操作系统级别的安全。应用软件的安全，包括对数据或业务功能的访问，在预期的安全情况下，操作者只能访问应用软件的特定功能、有限的数据等。操作系统级别的安全是确保只有具备系统平台访问权限的用户才能访问，包括对系统的登录或远程访问。

随着互联网和基于互联网的应用系统的不断发展，软件安全问题日益严重。导致软件出现安全问题的主要原因或根源是软件的安全漏洞。

11.2.1 安全漏洞的概念

安全漏洞特指硬件、软件、协议在逻辑设计上或具体实现上或系统安全策略上存在的缺陷或错误，这些缺陷或错误可以被不法者或者电脑黑客利用获得计算机系统的额外权限，在未授权或提高权限的情况下通过植入木马、病毒等方式来攻击或控制整个电脑，从而窃取电脑中的重要资料和信息，甚至破坏系统。具体举例来说，比如在 Intel Pentium 芯片中存在的逻辑错误，在 Sendmail 早期版本中的编程错误，在 NFS 协议中认证方式上的弱点，在 UNIX 系统管理员设置匿名 FTP 服务时配置不当的问题都可能被攻击者使用，威胁到系统的安全。因而这些都可以认为是系统中存在的安全漏洞。

安全漏洞一般是在系统具体实现和具体使用中产生的错误，但并不是系统中存在的错误都是安全漏洞。只有能威胁到系统安全的错误才是漏洞。许多错误在通常情况下并不会对系统安全造成危害，只有被人在某些条件下故意使用时才会影响系统安全。因此，安全漏洞是硬件、软件或使用策略上存在缺陷和不足，它们的存在会使计算机遭受病毒和黑客攻击，导致潜在的安全威胁。

漏洞的产生主要是由于程序员不正确和不安全编程引起的。大多数程序员在编程初始就没有考虑到安全问题。在后期，由于用户不正确地使用以及不恰当地配置都可导致漏洞的出现。归根求源，软件安全漏洞的存在是由于在软件开发的过程中对软件安全重视不够、过度追求按期完工引起的。分析漏洞产生原因，目的就在于希望从根本上减少漏洞的产生。

1. 安全漏洞产生的原因

通过分类、统计分析，多数漏洞产生的原因可归结为以下几种。

(1) 输入验证错误。漏洞的产生是由于未对用户提供的输入数据的合法性作适当的检查。这种错误导致的安全问题最多。

(2) 访问验证错误。漏洞的产生是由于程序的访问验证部分存在某些可利用的逻辑错误或用于验证的条件不足以确定用户的身份而造成的。这类缺陷使得非法用户绕过访问控制成为可能,从而导致未经授权的访问。

(3) 竞争条件错误。漏洞的产生是由于程序在处理文件等实体时在时序和同步方面存在问题,在处理的过程中可能存在一个机会窗口使攻击者能够施加外来的影响。

(4) 意外情况处置错误。漏洞的产生是由于程序在它的实现逻辑中没有考虑到一些本应该考虑的意外情况。这种错误是比较常见的。例如,在没有检查文件是否存在的情况下就直接打开设备文件而导致的拒绝服务;在没有检查文件是否存在的情况下就打开文件提取内容进行比较而导致的绕过验证;上下文攻击导致的执行任意代码等。

(5) 配置错误。漏洞的产生是由于系统和应用的配置有误。或是软件安装在错误的地方,或是参数配置错误,或是访问权限配置错误等。

(6) 环境错误。由一些环境变量的错误或恶意设置造成的漏洞,导致有问题的特权程序可能去执行攻击代码。

(7) 设计错误。这个类别是非常笼统的,严格来说,大多数漏洞的存在都是设计错误。

2. 安全漏洞的危害

漏洞的存在会对系统造成很严重的危害,因为它可能会被攻击者利用,继而破坏系统的安全特性,而它本身不会直接对系统造成危害。通常而言漏洞会对以下五种系统安全特性造成危害。

(1) 系统的完整性(Integrity)。攻击者可利用漏洞入侵系统,对系统数据进行非法篡改,从而达到破坏数据完整性的目的。

(2) 系统的可用性(Availability)。攻击者利用漏洞破坏系统或者网络的正常运行,导致信息或网络服务不可用,合法用户的正常服务要求得不到满足。

(3) 系统的机密性(Confidentiality)。攻击者利用漏洞给非授权的个人和实体泄漏受保护信息。有些时候,机密性和完整性是交叠的。

(4) 系统的可控性(Controlability)。攻击者利用漏洞对授权机构控制信息的机密性造成危害。

(5) 系统的可靠性(Reliability)。攻击者利用漏洞对用户认可的质量特性造成危害。

漏洞的危害是多方面的。近年来许多突发的、大规模的网络安全事件多数都是由于漏洞而导致的。

3. 安全漏洞的分类

出现软件故障现象的原因是软件存在漏洞。任何软件,不论它看起来是多么安全,其中都隐藏漏洞。软件安全的目的是尽可能消除软件漏洞,确保软件在恶意攻击下仍然正常运行。

软件安全问题加剧的三个原因是:①互联性(多数计算机与 Internet 相连,多数软件系统互联于 Internet);②可扩展性(通过接受更新或者扩展件使系统升级);③复杂性(代码行数增加、网络式、分布式)。

对于应用软件来说,它们的安全包括两个层面:①应用软件本身的安全(一般来说,应用软件的安全问题主要是由软件漏洞导致的,这些漏洞可以是设计上的缺陷或是编程上的问题,甚至是开发人员预留的后门);②应用软件的数据安全(包括数据存储安全和数据传输安全两个方面)。

我们已知道,软件漏洞就是攻击者可以开发和利用应用软件中存在的错误。包括从局部的执行错误(如使用 C/C++中调用 gets()函数),到程序间的接口错误(如在访问控制检查和

文件操作之间的竞争状态),甚至更高的设计层次错误(例如,不安全的设计导致的应用软件进入到出错处理或系统恢复程序,或者,包括错误地传递信用数据)。

尽管错误常被攻击者利用,但攻击者通常不关心漏洞是因为瑕疵或者错误。因为现在攻击正在变得越来越复杂,漏洞关系到的内容在不断改变。虽然包括著名的竞争状态在内的时序攻击几年以前还被认为是特殊的,但是现在它们非常普通。类似地,缓冲区溢出攻击早期是比较专业的手段,现在却出现更加高级的方法,如图 11-1 所示。

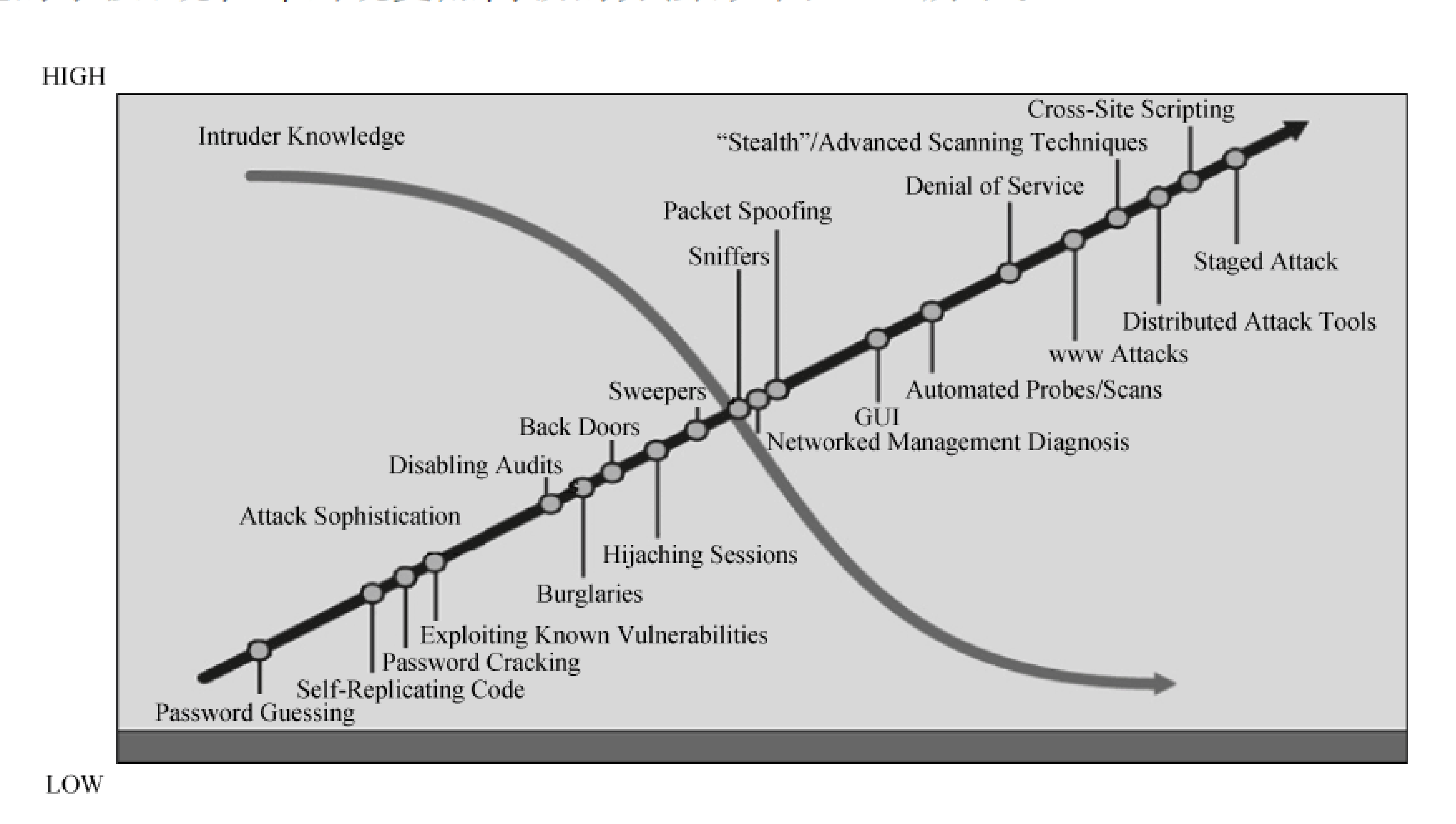


图 11-1 黑客攻击方式的进化

漏洞可以典型地分为两类——实现层面的错误和设计层面的错误。

1) 实现层面的错误

实现层面的错误或漏洞示例:2001 年的红色代码(Code Red)蠕虫利用 Microsoft 的 IIS Web 服务器的软件缺陷。它的字符串变量是 Unicode 字符类型(每个字符占用两个字节),在计算缓冲区的时候应该是偏移量为 2,但 IIS 在计算缓冲区大小的时候却按照偏移量为 1 错误地计算缓冲区,导致在 14 个小时内,就有 359 000 台机器传染了红色代码蠕虫。

在实现中几个常见的致命安全漏洞有缓冲区溢出、SQL 注入、跨站脚本。

(1) 缓冲区溢出示例。当一个程序允许输入的数据大于已分配的缓冲区大小时,缓冲区溢出就会发生。有些语言具有直接访问应用程序内存的能力,如果未能处理好用户的数据就会造成缓冲区溢出。C 和 C++是受缓冲区溢出影响的两种最常见的编程语言。缓冲区溢出造成的后果小到系统崩溃,大到攻击者获取应用程序的完全控制权。1988 年,第一个 Internet 蠕虫——Morris 蠕虫就是对 Finger 服务器进行攻击,造成缓冲区溢出,几乎导致 Internet 瘫痪。

(2) SQL 注入示例。通过把 SQL 命令插入到 Web 表单递交或输入域名或页面请求的查询字符串,最终达到欺骗服务器执行恶意的 SQL 命令。任何可以与数据库交互的编程语言都可能出现 SQL 注入漏洞。SQL 注入的最大的威胁是攻击者可以获得个人的隐私信息或敏感数据;也可能导致服务器甚至网络遭到入侵。

(3) 跨站脚本示例。跨站脚本(XSS,Cross-site Scripting)漏洞是一类专门针对 Web 应用程序的漏洞,它会使得产生漏洞的 Web 服务器绑定的用户数据(通常保存在 cookie 中)被泄露给恶意的第三方。所谓"跨站"是指;当一个客户端访问了可正常提供服务但是有漏洞的 Web 服务器后,cookie 从此客户端传递给了攻击者控制的站点。任何用于构建网站的编程语言或

者技术都可能受到影响。

（4）其他致命安全漏洞：格式化字符串、整数溢出、命令注入、未能处理错误信息、未能保护好网络流量、使用 Magic UPL 及隐藏表单字段、未能正确使用 SSL 和 TLS、使用基于弱口令的系统、未能安全地存储和保存数据、信息泄露、不恰当的文件访问、脆弱的网络域名安全、竞争条件、未认证的密钥交换、密码学强度随机数及不良可用性等。

2）设计层面的错误

设计层面的问题实例包括面向对象系统中的出错处理、对象共享和访问问题、未保护的数据通道（内部的和外部的）、不正确或缺失的访问控制机制、缺少审计或日志、不正确的日志、顺序错误和同步错误（特别是在多线程系统中）。这些缺陷几乎总是导致安全风险。

应用软件的安全总是涉及保护信息、服务，对手的技能和资源，以及潜在补救花费，是风险管理的任务。尤其是在设计层面的风险分析能帮助我们识别潜在的安全问题和它们的影响。软件风险一旦被识别和归类就有助于指导我们进行软件安全问题的防范。

11.2.2 网络安全相关技术

在当今信息社会、网络社会的时代，全球上的计算机都通过网络或 Internet 联到一起，信息安全或应用软件安全的内涵发生了根本的变化，应用软件的安全问题实质上可以认为是网络应用的安全问题。为了保障和提升网络应用的安全，我们需要应用漏洞扫描、防火墙、认证与加密、入侵检测、防病毒、访问控制、安全审计以及虚拟专用网络（VPN）等技术手段，这是因为很多网络应用的安全问题可以用技术手段来解决。

1. 漏洞扫描技术

漏洞扫描就是对计算机系统或者其他网络设备进行安全相关的检测以找出网络中存在的安全隐患和可能被黑客利用的漏洞，并针对每个具体漏洞给出一个详细的解决方案。

漏洞扫描技术是主动检查自身网络安全、发现问题、及时堵住漏洞，从而提高网络抗攻击能力的一种技术，是网络安全保障措施中不可或缺的一个环节。

2. 防火墙技术

防火墙（Firewall）技术是一种安全隔离技术，它通过在两个安全策略不同的网络之间设置防火墙来控制两个网络之间的互访行为。

防火墙实际上是一种隔离技术，它将内部网和公众访问网（Internet）分开，在两者之间设置一道屏障，防止来自不明入侵者的所有通信。

防火墙也是一种网络安全设备，它自身具有较强的抗攻击能力，它对两个或多个网络之间传输的数据包按照一定的安全策略来实施检查、过滤，以决定网络之间的通信是否被允许，并监视网络运行状态。

防火墙能够过滤进出网络的数据包，管理进出网络的访问行为，封堵某些禁止的访问行为，记录通过防火墙的信息内容和活动，对网络攻击进行检测和告警。

3. 认证与加密技术

信息加密是网络安全的有效策略之一。一个加密的网络，不但可以防止非授权用户的搭线窃听和入网，而且也是对付恶意软件的有效方法之一。

信息加密的目的是保护计算机网络内的数据、文件，以及用户自身的敏感信息。网络加密常用的方法有链路加密、端到端加密和节点加密三种。链路加密的目的是保护链路两端网络设备间的通信安全；节点加密的目的是对源节点计算机到目的节点计算机之间的信息传输提供保护；端到端加密的目的是对源端用户到目的端用户的应用系统通信提供保护。用户可以

根据需求酌情选择上述加密方式。

信息加密过程是通过各种加密算法实现的，目的是以尽量小的代价提供尽量高的安全保护。在大多数情况下，信息加密是保证信息在传输中的机密性的唯一方法。如果按照收发双方密钥是否相同来分类，可以将这些加密算法分为常规密钥算法和公开密钥算法。采用常规密钥方案加密时，收信方和发信方使用相同的密钥，即加密密钥必须通过安全的途径传送。因此，密钥管理成为系统安全的重要因素。采用公开密钥方案加密时，收集方和发信方使用的密钥互不相同，而且几乎不可能从加密密钥推导出解密密钥。公开密钥加密方案的优点是可以适应网络的开放性要求，密钥管理较为简单，尤其可方便地实现数字签名和验证。

加密策略虽然能够保证信息在网络传输的过程中不被非法读取，但是不能够解决在网络上通信的双方相互确认彼此身份的真实性问题。这需要采用认证策略解决。所谓认证，是指对用户的身份"验明正身"。目前在网络安全解决方案中，多采用两种认证形式，一种是第三方认证，另一种是直接认证。基于公开密钥框架结构的交换认证和认证的管理，是将网络用于电子政务、电子业务和电子商务的基本安全保障。它通过对受信用户颁发数字证书并且联网相互验证的方式，实现了对用户身份真实性的确认。

除了用户数字证书方案外，网络上的用户身份认证，还有针对用户账户名＋静态密码在使用过程中的脆弱性推出的动态密码认证系统，以及近年来正在迅速发展的各种利用人体生理特征研制的生物电子认证方法。另外，为了解决网络通信中信息的完整性和不可否认性，人们还使用了数字签名技术。

4. 入侵检测技术

"入侵"(Intrusion)是个广义的概念，不仅包括发起攻击的人(如恶意的黑客)取得超出合法范围的系统控制权，也包括收集漏洞信息，造成拒绝访问(Denial of Service)等对计算机系统造成危害的行为。

入侵检测(Intrusion Detection，ID)是对入侵行为的检测。它通过收集和分析网络行为、安全日志、审计数据、其他网络上可以获得的信息以及计算机系统中若干关键点的信息，检查网络或系统中是否存在违反安全策略的行为和被攻击的迹象。入侵检测作为一种积极主动的安全防护技术，提供了对内部攻击、外部攻击和误操作的实时保护，在网络系统受到危害之前拦截和响应入侵。因此被认为是防火墙之后的第二道安全闸门，在不影响网络性能的情况下能对网络进行监测。入侵检测通过执行以下任务来实现：监视、分析用户及系统活动；系统构造和弱点的审计；识别反映已知进攻的活动模式并向相关人士报警；异常行为模式的统计分析；评估重要系统和数据文件的完整性；操作系统的审计跟踪管理，并识别用户违反安全策略的行为。

5. 防病毒技术

根据《中华人民共和国计算机信息系统安全保护条例》，对计算机病毒的定义规定如下：计算机病毒，是指编制或者在计算机程序中插入的破坏计算机功能或者毁坏数据，影响计算机使用，并能自我复制的一组计算机指令或者程序代码。

计算机病毒的主要危害：直接破坏计算机数据信息，占用磁盘空间和对信息的破坏，抢占系统资源，影响计算机运行速度，计算机病毒错误与不可预见的危害，计算机病毒的兼容性对系统运行的影响，给用户造成严重的心理压力。

当前主要用到的防病毒技术是特征代码法、校验和法(将正常文件的内容，计算其校验和，将该校验和写入文件中或写入别的文件中保存。在文件使用过程中，定期地或每次使用文件前，检查文件现在内容算出的校验和与原来保存的校验和是否一致，因而可以发现文件是否感

染,这种方法叫校验和法)、行为监测法、启发式扫描及虚拟机技术等。

6. 虚拟专网技术

虚拟专网(Virtual Private Network,VPN),是通过一个公用网络(通常是因特网)建立一个临时、安全的连接,是一条穿过混乱的公用网络的安全、稳定的隧道。VPN 是对企业内部网的扩展,通过它可以帮助远程用户、公司分支机构、商业伙伴及供应商同公司的内部网建立可信的安全连接,并保证数据的安全传输。VPN 可用于不断增长的移动用户的全球因特网接入,以实现安全连接;可用于实现企业网站之间安全通信的虚拟专用线路,用于经济有效地连接到商业伙伴和用户的安全外联网虚拟专用网。

虚拟专网技术的核心是采用隧道技术,将内部网络的数据加密封装后,通过虚拟的公网隧道进行传输,从而防止敏感数据的被窃。

7. 上网行为管理技术

上网行为管理产品及技术是专用于防止非法信息恶意传播,避免国家机密、商业信息、科研成果泄漏的产品;并可实时监控、管理网络资源使用情况,提高整体工作效率。上网行为管理产品系列适用于需实施内容审计与行为监控、行为管理的网络环境,尤其是按等级进行计算机信息系统安全保护的相关单位或部门。

上网行为管理系统拥有上网行为审计与网络安全防护的双重应用功能。借助共享公共框架的系统平台,可以完成行为审计、内容审计、流量统计、内容监控、记录状态、系统安全管理、网页内容管理、邮件内容管理、IM&P2P、审计管理等具体应用。

11.2.3　解决软件安全问题的方法

软件安全涉及软件工程、编程语言、安全工程等。按照 Gary Mc Graw 建议,解决软件安全问题主要从应用风险管理、软件安全最佳实践以及知识这三方面着手。

1. 应用风险管理

安全就是风险管理。而风险就是在项目过程中有可能发生的某些意外事情。例如:缺少对开发工具的培训。

风险管理的关键是随着软件项目的展开,不断地确定和追踪风险。在这过程中,要预先识别、分析和修复那些对应用有负面影响的软件缺陷,并决定花多少成本去修复它。表 11-1 给出了某软件项目的风险管理计划。

表 11-1　某软件项目风险管理计划列表

项目风险管理计划					
风险识别		风险评估		风险应对措施	
潜在风险	后果	可能性	严重性	应急措施	预防措施
客户的需求不明确	客户不接受产品或拒绝付款	5	9	按照客户的要求修改	事先进行需求评审
项目范围定义不清楚	项目没完没了	8	9	按照客户要求变更	事先定义清楚并由客户确认
项目目标不明确	项目进度拖延或成本超支	6	8	修改项目目标	实现明确项目目标
与客户沟通不够	软件不能满足客户需求	5	9	立即与客户进行沟通	制定沟通管理计划
…	…	…	…	…	…

2. 软件安全的最佳实践

软件安全从业者完成许多不同的工作来管理软件安全风险，包括制作、滥用或误用安全案例，列出标准的安全需求，执行体系结构风险分析，建立基于风险的安全测试计划，使用静态分析工具，执行安全测试，在最终环境中执行渗透测试以及清理安全漏洞。

事实上，软件安全的最佳实践涉及七个方面的活动：安全需求分析、用例（Use Case）滥用、体系结构风险分析、代码审核、渗透测试、基于风险的安全测试以及安全操作。图 11-2 给出了这七项活动在软件开发生命周期中的最佳介入点。

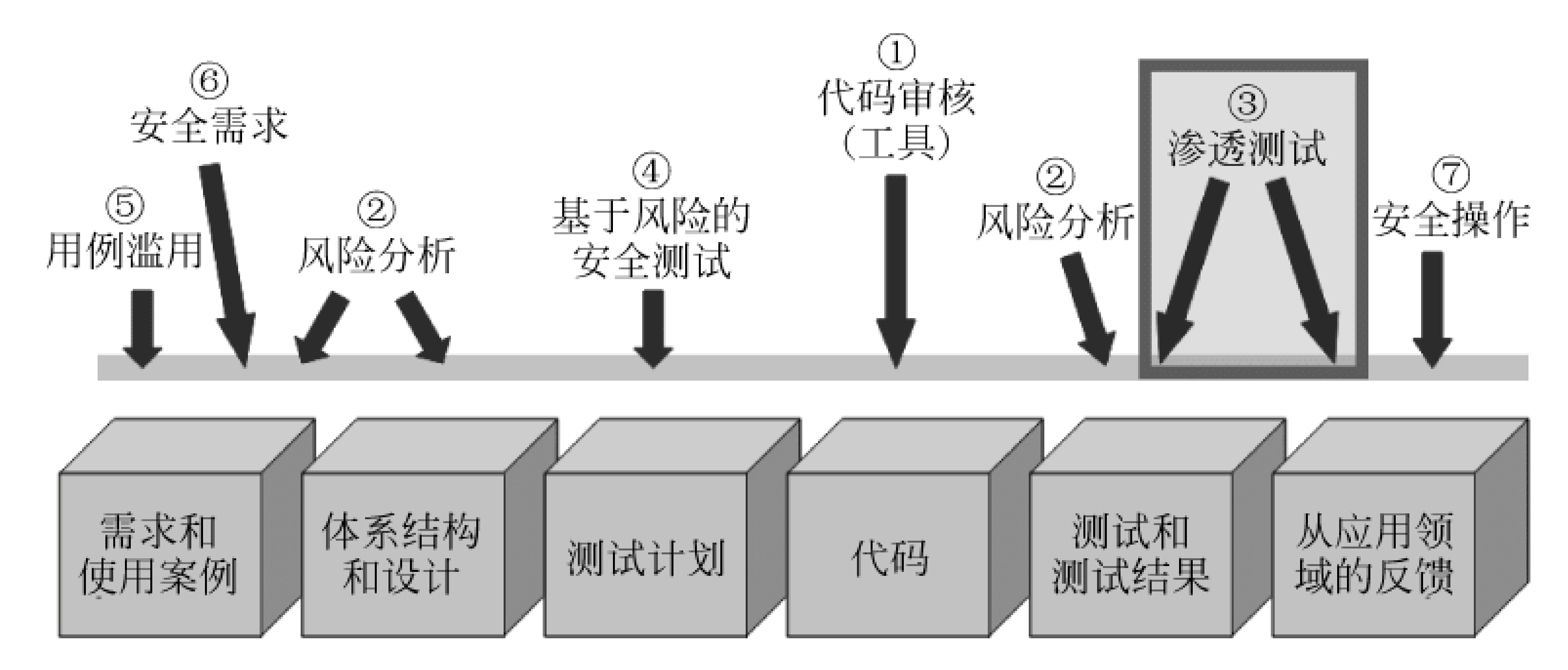


图 11-2 软件安全的最佳实践活动图

3. 知识

软件安全最大的挑战是我们缺乏在软件安全方面的知识和经验。因此我们需要，也非常有必要给我们的开发人员、设计人员、分析人员更多的有关软件安全的知识的培训，以期他们具备丰富的软件安全的知识。

要教育大家软件安全人人有责：①开发人员必须实施安全工程，保证建造的系统是可防御的而非布满漏洞。②软件操作人员必须继续架构合理的网络，保护它们并维持它们的运行。③管理员必须理解现代系统的分布式本质，并开始实施最低特权原则。④用户必须认识到软件是可以安全的，所以可以与软件供应商合作，共享软件的价值。还必须认识到，在任何设计中用户都是最后的防御阵地。⑤软件经理主管人员必须认识到，在安全设计和安全分析上尽早投资会有助于提高用户对他们的产品的信任程度。

11.2.4 软件安全测试方法与技术

加密与认证、访问控制等安全技术在软件安全中发挥着关键作用，安全本身是整个系统的一个凸显特性，不仅仅是安全机制和安全技术的应用。不管是缓冲区或是无关紧要的图形用户界面，它们都存在着安全问题。我们常说，软件系统的安全必须能够经受住正面的攻击，以及侧面的和背后的攻击。

普通的软件测试的主要目的是：确保软件不会去完成没有预先设计的功能，确保软件能够完成预先设计的功能。但是，安全测试更有针对性，同时可能采用一些和普通测试不一样的测试手段，如攻击和反攻击技术。因此，实际上，安全测试实际上就是一轮多角度、全方位的攻击和反攻击，其目的就是要抢在攻击者之前尽可能多地找到软件中的漏洞，以减少软件遭到攻击的可能性。

安全测试必须采用两种不同的方法：①测试安全机制来确保它们的功能被正确实现；②明白和模拟攻击者的途径，促进基于风险的安全测试。

1. 软件安全测试

一般来说，对安全要求不高的软件，其安全测试可以混在单元测试、集成测试、系统测试里

一起做。但对安全有较高需求的软件，则必须做专门的安全测试，以便在破坏之前预防并识别软件的安全问题。

1）安全测试定义

安全测试(Security Testing)是用来验证集成在软件内的保护机制是否能够在实际中保护系统免受非法的侵入，在测试软件系统中对程序的危险防止和危险处理进行的有效性测试和验证，它是有关验证应用软件的安全等级和识别潜在安全缺陷的过程。

软件安全测试的主要目的是查找软件自身程序设计中存在的安全隐患，并检查软件对非法侵入的防范能力。不同的安全指标其测试策略是不同的。注意：安全测试并不最终证明应用软件是安全的，而是用于验证所设立策略的有效性，这些对策是基于威胁分析阶段所做的假设而选择的。例如，测试应用软件在防止非授权的内部或外部用户的访问或故意破坏等情况时的运作。

2）安全测试工作

安全测试一般要做的工作有：①全面检验软件在软件需求规格说明中规定的防止危险状态措施的有效性和在每一个危险状态下的处理反应情况；②对软件设计中用于提高安全性的逻辑结构、处理方案，进行针对性测试；③在异常条件下测试软件，以表明不会因可能的单个或多个输入错误而导致不安全状态；④用错误的安全性关键操作进行测试，以验证系统对这些操作错误的反应；⑤对安全性关键的软件单元功能模块要单独进行加强的测试以确认其满足安全性需求。

3）安全测试方法

有许多的测试手段可以进行安全测试，目前主要安全测试方法有：

(1) 静态的代码安全测试。主要通过对源代码进行安全扫描，根据程序中数据流、控制流、语义等信息与其特有软件安全规则库进行匹对，从中找出代码中潜在的安全漏洞。静态的源代码安全测试是非常有用的方法，它可以在编码阶段找出所有可能存在安全风险的代码，这样开发人员可以在早期解决潜在的安全问题。正因为如此，静态代码测试比较适用于早期的代码开发阶段，而不是测试阶段。

(2) 动态的渗透测试。渗透测试也是常用的安全测试方法。是使用自动化工具或者人工的方法模拟黑客的输入，对应用系统进行攻击性测试，从中找出运行时刻所存在的安全漏洞。这种测试的特点就是真实有效，一般找出来的问题都是正确的，也是较为严重的。但渗透测试一个致命的缺点是模拟的测试数据只能到达有限的测试点，覆盖率很低。

(3) 程序数据扫描。一个有高安全需求的软件，在运行过程中数据是不能遭到破坏的，否则就会导致缓冲区溢出类型的攻击。数据扫描的手段通常是进行内存测试，内存测试可以发现许多诸如缓冲区溢出之类的漏洞，而这类漏洞使用除此之外的测试手段都难以发现。例如，对软件运行时的内存信息进行扫描，看是否存在一些导致隐患的信息，当然这需要专门的工具来进行验证，手工做是比较困难的。

4）安全测试内容

安全测试内容很多，但主要测试内容有：

(1) 功能验证。功能验证是采用软件测试当中的“黑盒”测试方法，对涉及安全的软件功能，如用户管理模块、权限管理、加密系统、认证系统等进行测试，主要验证上述功能是否有效。对安全的功能验证可以采用与一般的程序功能测试相似的方法，如“黑盒”测试方法、“白盒”测试方法或“灰盒”测试方法等用例来进行测试。

(2) 漏洞扫描。安全漏洞扫描主要是借助于特定的漏洞扫描器完成的。通过使用漏洞扫

描器,系统管理员能够发现系统存在的安全漏洞,从而在系统安全中及时修补漏洞。一般漏洞扫描分为两种类型:①主机漏洞扫描器是指在系统本地运行检测系统漏洞的程序;②网络漏洞扫描器是指基于网络远程检测目标网络和主机系统漏洞的程序。

(3) 模拟攻击。对于安全测试来说,模拟攻击测试是一组特殊的极端的测试方法,我们以模拟攻击来验证软件系统的安全防护能力。模拟攻击主要的攻击技术分为:①服务拒绝型攻击(企图通过使服务器崩溃来阻止提供服务,是最容易实施的攻击行为);②漏洞木马型攻击(主要是由于系统使用者已知系统漏洞未及时打补丁或者不小心安放了木马等原因导致的非法入侵行为);③伪装欺骗型攻击(这类攻击是使目标配置不正确的消息)。

(4) 侦听技术,也称网络监听。可以获取网络上传输的信息,而这些信息并不是发给自己的。网络侦听技术是一个常用的手段。可以有效地管理网络、诊断网络问题、检查网络的安全威胁。目前网络侦听工具有多种,有硬件,也有软件的。测评人员为了评测信息系统的安全环境,熟悉网络侦听技术。使用侦听技术是一种有效的安全测试方法。

2. 安全测试的主要测试手段——渗透测试

渗透测试(Penetration Testing)是完全模拟黑客可能使用的攻击技术和漏洞发现技术,对目标系统(软件)的安全做深入的探测,发现系统最脆弱的环节。它能直观地展现系统(软件)所面临的安全问题。

1) 渗透测试的必要性

渗透测试利用网络安全扫描器、专用安全测试工具和富有经验的安全工程师的人工经验对网络中的核心服务器及重要的网络设备,包括服务器、网络设备、防火墙等进行非破坏性质的模拟黑客攻击,目的是侵入系统,获取机密信息并将入侵的过程和细节产生报告给用户。

渗透测试和工具扫描可以很好地互相补充。工具扫描具有很好的效率和速度,但是存在一定的误报率和漏报率,并且不能发现高层次、复杂并且相互关联的安全问题;渗透测试需要投入的人力资源较大,对测试者的专业技能要求很高(渗透测试报告的价值直接依赖于测试者的专业技能),但是非常准确,可以发现逻辑性更强、更深层次的弱点。

2) 渗透测试分类

渗透测试涉及的内容很多、面很广,因此对它的分类有不同的侧重点。

(1) 根据渗透方法分类有:①"黑盒"测试(Zero-knowledge Testing,渗透者完全处于对系统一无所知的状态。通常,这种类型的测试,最初的信息获取来自 DNS、Web、E-mail 及各种公开对外的服务器);②"白盒"测试(测试者可以通过正常渠道向被测单位取得各种资料,包括网络拓扑、员工资料甚至网站或程序的代码片段,也能与单位其他员工进行面对面的沟通。这类的测试目的是模拟企业内部雇员的越权操作);③隐秘测试(隐秘是针对被测单位而言的。通常,接受渗透测试的单位网络管理部门会收到通知,在某些时间段进行测试。因此能够检测网络中出现的变化。但在隐秘测试中,被测单位也仅有极少数人知晓测试的存在,因此能够有效地检验单位中的信息安全事件监控、响应及恢复是否做得到位)。

(2) 根据渗透目标分类有:①主机操作系统渗透(对 Windows、Solaris、AIX、Linux、SCO、SGI 等操作系统本身进行渗透测试);②数据库系统渗透(对 MS-SQL、Oracle、MySql、Infomix、Sybase、DB2 等数据库应用系统进行渗透测试);③应用系统渗透(对渗透目标提供的各种应用,如 ASP、CGI、JSP、PHP 等组成的 WWW 应用进行渗透测试);④网络设备渗透(对各种防火墙、入侵检测系统、网络设备进行渗透测试)。

3) 攻方主要用到的渗透测试

攻方主要用到的渗透测试有以下几种。

(1) 内网测试。渗透测试人员由内部网络发起测试。这类测试能够模拟企业内部违规操作者的行为,其优势是绕过了防火墙的保护。内网测试常用的渗透方式有远程缓冲区溢出、口令猜测以及 B/S 或 C/S 应用程序测试。

(2) 外网测试。渗透测试人员完全处于外部网络,模拟对内部状态一无所知的外部攻击者的行为。包括对网络设备的远程攻击、口令管理安全性测试、防火墙规则试探与规避、Web 及其他开放应用服务的安全性测试。

(3) 不同网段/虚拟局域网之间的渗透测试。从某内/外部网段,尝试对另一网段/虚拟局域网进行渗透。通常可能用到的技术包括:对网络设备的远程攻击;对防火墙的远程攻击或规则探测、规避尝试。信息的收集和分析伴随着每一个渗透测试步骤,每一个步骤又有三个组成部分:操作、响应和结果分析。

(4) 端口扫描。通过对目标地址的 TCP 或 UDP 端口扫描,确定其所开放的服务的数量和类型,这是所有渗透测试的基础。通过端口扫描,可以基本确定一个系统的基本信息,结合安全工程师的经验可以确定其可能存在,以及被利用的安全弱点,为进行深层次的渗透提供依据。

(5) 远程溢出。这是当前出现的频率最高、威胁最严重,同时又是最容易实现的一种渗透方法,一个具有一般网络知识的入侵者就可以在很短的时间内利用现成的工具实现远程溢出攻击。对于防火墙内的系统同样存在这样的风险,只要对跨接防火墙内外的一台主机攻击成功,那么通过这台主机对防火墙内的主机进行攻击就易如反掌。

(6) 口令猜测。口令猜测也是一种出现概率很高的风险,几乎不需要任何攻击工具,利用一个简单的暴力攻击程序和一个比较完善的字典,就可以猜测口令。对一个系统账号的猜测通常包括对用户名的猜测和对密码的猜测两个方面。

(7) 本地溢出。指在拥有了一个普通用户的账号之后,通过一段特殊的指令代码获得管理员权限的方法。前提是首先要获得一个普通用户密码。也就是说由于导致本地溢出的一个关键条件是设置不当的密码策略。多年的实践证明,在使用经过前期的口令猜测阶段获取的普通账号登录系统之后,对系统实施本地溢出攻击,就能获取不进行主动安全防御的系统的控制管理权限。

(8) 脚本及应用测试。专门针对 Web 及数据库服务器进行。据最新技术统计,脚本安全弱点为当前 Web 系统,尤其是存在动态内容的 Web 系统比较严重的安全弱点之一。利用脚本相关弱点轻则可以获取系统其他目录的访问权限,重则将有可能取得系统的控制权限。因此对于含有动态页面的 Web、数据库等系统,Web 脚本及应用测试将是必不可少的一个环节。

(9) 无线测试。中国的无线网络还处于建设时期,但是由于无线网络的部署简易,在一些大城市的普及率已经很高了。北京和上海的商务区至少 80%的地方都可以找到接入点。通过对无线网络的测试,可以判断企业局域网安全性,已经成为越来越重要的渗透测试环节。

(10) 除了前述的测试手段外,还有一些可能会在渗透测试过程中使用的技术,包括社交工程学、拒绝服务攻击,以及中间人攻击。

4) 防守对渗透测试的关注

当具备渗透测试攻击经验的人们站到系统管理员的角度,要保障一个大网的安全时,我们会发现,需要关注的问题是完全不同的:从攻方的视角看,是“攻其一点,不及其余”,只要找到一点漏洞,就有可能撕开整条战线;但从守方的视角看,却发现往往“千里之堤,毁于蚁穴”。因此,需要有好的理论指引,从技术到管理都注重安全,才能使网络固若金汤。

3. 软件安全测试过程

软件安全测试过程在安全测试中分为两种测试过程:反向安全测试过程和正向安全测试

过程。

1）反向安全测试过程

大部分软件的安全测试都是依据缺陷空间反向设计原则来进行的，即事先检查哪些地方可能存在安全隐患，然后针对这些可能的隐患进行测试。因此，反向测试过程是从缺陷空间出发，建立缺陷威胁模型，通过威胁模型来寻找入侵点，对入侵点进行已知漏洞的扫描测试。好处是可以对已知的缺陷进行分析，避免软件里存在已知类型的缺陷，但是对未知的攻击手段和方法通常会无能为力。

2）正向安全测试过程

为了规避反向设计原则所带来的测试不完备性，需要一种正向的测试方法来对软件进行比较完备的测试，使测试过的软件能够预防未知的攻击。

（1）标识测试空间。对测试空间的所有的可变数据进行标识，由于进行安全测试的代价高昂，其中要重点对外部输入层进行标识。例如，需求分析、概要设计、详细设计、编码这几个阶段都要对测试空间进行标识，并建立测试空间跟踪矩阵。

（2）精确定义设计空间。重点审查需求中对设计空间是否有明确定义，和需求牵涉到的数据是否都标识出了它的合法取值范围。在这个步骤中，最需要注意的是“精确”二字，要严格按照安全原则来对设计空间做精确的定义。

（3）标识安全隐患。根据找出的测试空间和设计空间以及它们之间的转换规则，标识出哪些测试空间和哪些转换规则可能存在安全隐患。例如，测试空间愈复杂，即测试空间划分越复杂或可变数据组合关系越多也越不安全。还有转换规则愈复杂，则出问题的可能性也愈大，这些都属于安全隐患。

（4）建立和验证入侵矩阵。安全隐患标识完成后，就可以根据标识出来的安全隐患建立入侵矩阵。列出潜在安全隐患，标识出存在潜在安全隐患的可变数据，和标识出安全隐患的等级。其中对于那些安全隐患等级高的可变数据，必须进行详尽的测试用例设计。

3）正向和反向测试的区别

正向测试过程是以测试空间为依据寻找缺陷和漏洞，反向测试过程则是以已知的缺陷空间为依据去寻找软件中是否会发生同样的缺陷和漏洞，两者各有其优缺点。反向测试过程主要的一个优点是成本较低，只要验证已知的可能发生的缺陷即可，但缺点是测试不完善，无法将测试空间覆盖完整，无法发现未知的攻击手段。正向测试过程的优点是测试比较充分，但工作量相对来说较大。因此，对安全要求较低的软件，一般按反向测试过程来测试即可，对于安全要求较高的软件，应以正向测试过程为主，反向测试过程为辅。

4. 安全测试主要关注的问题

软件的安全有很多方面的内容，主要的安全问题是由软件本身的漏洞造成的，通常安全测试关注的缺陷和漏洞有如下几个方面。

1）缓冲区溢出

缓冲区溢出已成为软件安全的头号公敌，许多实际中的安全问题都与它有关。造成缓冲区溢出问题通常有以下两种原因。

（1）设计空间的转换规则的校验问题。即缺乏对可测数据的校验，导致非法数据没有在外部输入层被检查出来并丢弃。非法数据进入接口层和实现层后，由于它超出了接口层和实现层的对应测试空间或设计空间的范围，从而引起溢出。

（2）局部测试空间和设计空间不足。当合法数据进入后，由于程序实现层内对应的测试空间或设计空间不足，导致程序处理时出现溢出。

2）加密缺陷

如下几个加密缺陷是不安全的。

(1) 使用不安全的加密算法。加密算法强度不够，一些加密算法甚至可以用穷举法破解。

(2) 加密数据时密码是由伪随机算法产生的，而产生伪随机数的方法存在缺陷，使密码很容易被破解。

(3) 身份验证算法存在缺陷。

(4) 客户机和服务器时钟未同步，给攻击者足够的时间来破解密码或修改数据。

(5) 未对加密数据进行签名，导致攻击者可以篡改数据。所以，对于加密进行测试时，必须针对这些可能存在的加密弱点进行测试。

3）错误处理

一般情况下，错误处理都会返回一些信息给用户，返回的出错信息可能会被恶意用户利用来进行攻击，恶意用户能够通过分析返回的错误信息知道下一步要如何做才能使攻击成功。如果错误处理时调用了一些不该有的功能，那么错误处理的过程将被利用。错误处理属于异常空间内的处理问题，异常空间内的处理要尽量简单，使用这条原则来设计可以避免这个问题。但错误处理往往牵涉到易用性方面的问题，如果错误处理的提示信息过于简单，用户可能会一头雾水，不知道下一步该怎么操作。所以，在考虑错误处理的安全性的同时，需要和易用性一起进行权衡。

4）权限过大

如果赋予过大的权限，就可能导致只有普通用户权限的恶意用户利用过大的权限做出危害安全的操作。例如没有对能操作的内容做出限制，就可能导致用户可以访问超出规定范围的其他资源。进行安全测试时必须测试应用程序是否使用了过大的权限，重点要分析在各种情况下应该有的权限，然后检查实际中是否超出了给定的权限。权限过大问题本质上属于设计空间过大问题，所以在设计时要控制好设计空间，避免设计空间过大造成权限过大的问题。

5. 如何做好安全测试

许多软件安全测试经验告诉我们，做好软件安全测试的必要条件是：①充分了解软件安全漏洞；②评估安全风险；③拥有高效的软件安全测试技术和工具。

1）充分了解软件安全漏洞

评估一个软件系统的安全程度，需要从设计、实现和部署三个环节同时着手。通常评估软件系统安全的方法是：①要确定软件产品对应的防范要求(Protection Profile，PP)。一个PP定义了一类软件产品的安全特性模板，例如数据库的PP、防火墙的PP等。②根据PP再提出具体的安全功能需求，如用户的身份认证实现。③确定安全对象以及是如何满足对应的安全功能需求的。因此，一个安全软件的三个环节，哪个出问题都不行。

2）安全测试的评估

做完安全测试后，软件是否能够达到预期的安全程度，这是安全测试人员最关心的问题，因此需要建立对测试后的安全评估机制。一般从以下两个方面进行评估。

(1) 安全缺陷数据评估。如果发现软件的安全缺陷和漏洞越多，可能遗留的缺陷也越多。进行这类评估时，必须建立基线数据作为参照，否则评估起来没有依据就无法得到正确的结论。

(2) 采用漏洞植入法来进行评估。漏洞植入法和可靠性测试里的故障插入测试是同一道理，只不过这里是在软件里插入一些有安全隐患的问题。采用漏洞植入法时，先让不参加安全测试的特定人员在软件中预先植入一定数量的漏洞，最后测试完后看有多少植入的漏洞被发

现,以此来评估软件的安全测试做得是否充分。

3) 采用安全测试技术和工具

可使用专业的具有特定功能的安全扫描软件来寻找潜在的漏洞,将已经发生的缺陷纳入缺陷库,然后通过自动化测试方法来使用自动化缺陷库进行轰炸测试。例如,使用一些能够模拟各种攻击的软件来进行测试。

11.3 应用软件安全性/安全测试工具

当前,应用软件安全性(Safety)测试工具选型比较困难,没有那种让大家很满意,能够满足安全测试各方面要求的工具;而应用软件安全(Security)测试工具比较成熟的商用工具较多,如 Microsoft Threat Analysis & Modeling、IBM Rational AppScan 及 HP Fortify 等。

11.3.1 IBM Rational AppScan

IBM Rational AppScan 是一个 Web 应用安全测试工具包,也是唯一一个在所有级别应用上提供全面纠正任务的工具。

AppScan 扫描 Web 应用的基础架构,进行安全漏洞测试并提供可行的报告和建议。AppScan 对扫描能力、零时差补丁升级、配置向导和详细的报表系统等都进行了整合,简化使用,增强用户效率,有利于安全防范和保护 Web 应用基础架构。

AppScan 其实是一个产品家族,包括众多的应用安全扫描产品,从开发阶段的源代码扫描的 AppScan Source Edition,到针对 Web 应用进行快速扫描的 AppScan Standard Edition,以及进行安全管理和汇总整合的 AppScan Enterprise Edition 等。我们经常说的 AppScan 就是指的桌面版本的 AppScan,即 AppScan Standard Edition。其安装在 Windows 操作系统上,可以对网站等 Web 应用进行自动化的应用安全扫描和测试。

11.3.2 JSky

JSky(中文名字为竭思),是深圳市宇造诺赛科技有限公司的产品,是一款简明易用的 Web 漏洞扫描软件,它是针对网站漏洞扫描的安全软件,JSky 能够评估一个网站是否安全,对网站漏洞进行分析,判断是否存在漏洞,又称为网站漏洞扫描工具。

JSky 作为一款国内著名的网站漏洞扫描工具,提供网站漏洞扫描服务,即能查找出网站中的漏洞;网站漏洞检测工具提供网站漏洞检测服务,即能模拟黑客攻击来评估计算机网站安全。渗透测试模块能模拟黑客攻击,让您立刻知道问题的严重性。

有了 JSky 之后,网站管理者就可以方便快捷地进行网站漏洞分析,然后进行网站漏洞修复,这样就能减少网站被攻击的危害,保证公司正常业务的开展,维持企业的形象。

11.3.3 WebPecker

WebPecker 是一款小巧、实用的网站安全检测工具,又名网站啄木鸟。WebPecker 能很好地帮助测试人员检测本地漏洞、恶意网站、SQL 注入、网站管理后台漏洞等,可以测试正在浏览的网站、网页中是否存在木马,提醒用户防止他的重要信息泄露或丢失。

WebPecker 网站啄木鸟核心技术优势如下。

(1) SQL 注入网页抓取。WebPecker 的网页抓取模块采用广度优先爬虫技术以及网站目录还原技术。广度优先的爬虫技术不会产生爬虫陷入的问题,网站目录还原技术则去除了

无关结果，提高抓取效率。

(2) SQL注入状态扫描技术(非错误检测)。WebPecker不同于传统的针对错误反馈判断是否存在注入漏洞的方式，而采用自主创新的状态检测来判断。所谓状态检测，即：针对某一链接输入不同的参数，通过对网站反馈的结果使用向量比较算法进行比对判断，从而确定该链接是否为注入点，此方法不依赖于特定的数据库类型、设置以及CGI语言的种类，对于注入点检测全面，不会产生漏报现象。而常见的SQL注入扫描产品均不具备此项技术。

(3) 注入验证基于注入状态。WebPecker采用状态检测来对数据库的数据进行猜解，无论网站采用什么CGI语言，无论网站是否反馈错误信息，都能进行正常的猜解，而常见的SQL注入扫描产品均不具备此项技术。

习题

1. 什么是软件安全性？各种安全性定义的共同点有哪些？简述软件安全性工作内容。
2. 安全性分析包括哪些内容？如何开展安全性分析？
3. 安全性测试有哪些方法和技术？如何开展安全性测试？
4. 什么是软件安全？什么是安全漏洞？简述安全漏洞的种类和产生的原因。
5. 哪些技术可以保障网络的安全？对这些技术进行简要阐述。
6. 应用软件安全测试涉及哪些方面的内容？主要关注哪些问题？简述渗透测试的基本方法。



第12章 软件国际化与本地化测试

随着全球软件市场的急剧扩大,本地化的国际市场对于世界各国的软件企业来说越来越重要。越来越多的国内软件企业也开始认识到开发本地化市场的重要性,不过相对于一些国际软件巨头,比如 Microsoft、IBM 等来讲仍然是刚刚起步。

很多企业现在都同时推出国内版本和多国语言的本地化版本,这样做对于一个软件企业来说有很多的好处,具体包括:①增加企业利润。当软件企业开发一个应用软件的时候,也许最初它仅仅局限于国内的市场,可是如果附加一部分很少的投资,将它推向其他国家的市场,则可以利用较少的投资获得丰厚的回报。②增加市场份额。一个国家国内的市场份额总是有限的,可是国际大市场要比国内市场广阔得多,推出本地化的软件可以将自己的软件产品所占有的份额提高好几倍,甚至几十倍。③提高产品质量。国内、国际市场产品的质量可以互为提高,以提高企业的整体竞争力。

世界上的很多公司,如 Microsoft、HP、Novel、IBM 等都通过加速进入本地化市场获得了巨大的回报。

软件本地化是将一个软件产品按特定国家、地区或语言市场的需要进行加工,使之满足特定市场上的用户对语言和文化的特殊要求的软件生产活动。对于一个国际性的软件企业来说,其本地化工作是一个系统工程,它需要经历国际化、本地化、测试等多个步骤。

12.1 软件国际化与本地化

如今,市场的全球化,网络技术的广泛应用,计算机深入普及家庭。因此,开发适合于国际用户方便使用的应用程序的需求也日益增长。在这当中,软件的图形用户界面能够满足不同国家的不同文字输入、处理和显示成为软件国际化的基本要求。

12.1.1 软件国际化及本地化概念

通常软件在最初开发时,只有英文版本,根据需要,作者再把软件界面和文档翻译成不同国家、地区的语言版本。但是由于实现翻译的途径、翻译的工作效率、翻译的可重用性等因素各不相同,使翻译工作面临很大困境,也阻碍了软件的推广和应用。为了方便地将软件翻译成不同语言的版本,就需要一套翻译规范和通用工具,这就导致了“国际化”机制的出现。仅仅翻译是不够的,同一种语言在不同国家、地区可能存在多个支系,它们在表达习惯、语法结构甚至文字种类和编码上都有不同,方言更是千奇百怪,通用的翻译其质量肯定是不高的。涉及计算机领域,还存在操作习惯上的差别,而且对某种语言提供完美的输入、显示、打印、保存、传输并非一件轻而易举的事,这就导致了“本地化”机制的出现。

简而言之,“国际化”是“本地化”的一部分,主要是指国际化的实现机制和翻译工作,“本地

化”包含“国际化”，是对“国际化”的补充和完善，它还包括为实现对某种特定语言良好的支持而进行的有针对性的翻译调整以及对软件进行的打补丁工作。

“国际化”及“本地化”的国际组织（即 I18N 和 L10N 的国际组织）是 Openi18n，它原来是制定 GNU/Linux 自由操作系统上软件全球化标准的国际计划，后来扩充到 GNU/Linux 之外所有开放源代码的技术领域，因而更名为 Open Internationalization Initiative，由非营利组织 Free Standards Group 赞助，并为世界各大厂商所支持，对于 GNU/Linux 系统上的多国语言文字处理技术和环境有决定性的影响。各个开源软件开发组织通常都有负责“国际化”和“本地化”工作的分支机构。

1．软件国际化定义

软件国际化是全球化的产物。随着国际交流的密切和行业标准的国际性统一，一些大型软件或者热门软件，不但要提供一国语言的版本，还要提供其他国家语言的版本，这就是通常意义上的软件国际化。

国际化的英文单词是 Internationalization(I18n)，其定义是：软件国际化是在软件设计和文档开发过程中，使得功能和代码设计能处理多种语言和文化习俗，能够在创建不同语言版本时，不需要重新设计源程序代码的软件工程方法。

1）软件国际化的设计要求

软件国际化设计要遵循以下的通用准则：

（1）在国际化软件项目的初期融入国际化思想，并使国际化贯穿于项目的整个生命周期；

（2）采用单一源文件进行多语言版本的本地化，不针对不同的语言编写多套代码；

（3）需要本地化的文字与软件源代码分离，存储在单独的资源文件中；

（4）软件代码支持处理单字节字符集和多字节字符集文字的输入、输出和显示，并且遵守竖排和折行规则；

（5）软件代码应该支持 Unicode 标准，或者可以在 Unicode 和其他代码页（Code Page）互换；

（6）软件代码不要嵌入字体名，也不要假设使用某种字体；

（7）使用通用的图标和位图，避免不同区域的文化和传统差异，避免在图标和位图中嵌入需要本地化的文字；

（8）菜单、对话框等界面布局能够满足处理本地化文字的长度扩展的需要；

（9）源语言的文字要准确精简，使用一致的术语，避免歧义和拼写错误，以便进行本地化翻译；

（10）保证不同区域的键盘布局都能使用原软件的快捷键；

（11）考虑不同区域的法律和文化习俗对软件的要求；

（12）如果软件中采用第三方开发的软件或组件，需要检查和确认是否满足国际化的要求；

（13）保证源语言软件可以在不同的区域和操作系统上正确运行；

（14）软件代码中避免“硬编码”，不使用基于源语言的数字常量、屏幕位置、文件和路径名；

（15）字符串的缓冲区长度要满足本地化字符扩展的长度；

（16）软件能正确支持区域排序和大小写转换。

2）软件国际化的测试

软件产品的国际化是赋予软件产品的一种能力，这种能力可以使软件产品的本地化非常

容易。理想情况下，将国际化的软件本地化时，不需要修改源代码，只需要翻译资源库，进行特定的设置和定制就可以了。因此，软件国际化测试的目的是测试软件的国际化支持能力，发现软件国际化的潜在问题，保证软件在世界不同区域中都能正常运行。

在完成基本的语言版本(base code)测试之后，我们还要进行国际化、本地化的测试。通常情况下，国际化测试可以和基本语言版本测试同时进行，而本地化测试是在国际化版本测试完成之后进行的。

2. 软件本地化定义

本地化的英文单词是 Localization(L10N)，其定义是：将一个软件产品按特定国家/地区或语言市场的需要进行加工，使之满足特定市场上的用户对语言和文化的特殊要求的软件生产活动。包括翻译、重新设计、功能调整、功能测试以及是否符合当地的习俗、文化背景、语言和方言的验证等。

本地化是为解决网站、软件以及文档资料向其他国家推广时遇到的语言障碍问题。

(1) 网站本地化，即网站需要翻译成不同国家的语言，以便不同国家的人能够无障碍地阅读网站内容；

(2) 软件本地化，以便能够在目标国家推广；

(3) 将网站或软件本地化为全世界所有语种是不现实的，一般的惯例是只面向几种主要的语种(尤其是英语)进行本地化，如现在许多国内网站都有中英文两个版本。

本地化涉及区域(Locale)问题，即场所、本地。从地理上说，区域是某个地方(国家或地区)；是由语言、国家/地区，以及文化传统确定的用户环境特征集合，它决定了排列顺序、键盘布局，以及日期、时间、数字和货币格式等的通用设置。

1) 软件国际化和本地化的关系

国际化是为了解决软件能在各个不同语言、不同风俗的国家和地区使用的问题，对计算机设计和编程做出的某些规定。国际化是本地化的前提和基础，而本地化是国际化向特定本地语言环境的转换，本地化要适应国际化的规定。

2) 软件本地化的内容

软件本地化的内容包括软件用户界面、联机文档、组合键设置、度量衡和时区等。

3) 软件本地化的基本步骤

软件本地化的基本步骤是：①建立一个配置管理体系，跟踪目标语言各个版本的源代码；②创造和维护术语表；③从源语言代码中分离资源文件，或提取需要本地化的文本；④把分离或提取的文本、图片等翻译成目标语言；⑤把翻译好的文本、图片重新插入目标语言的源代码版本中；⑥如果需要，编译目标语言的源代码；⑦测试翻译后的软件，调整用户界面 UI 以适应翻译后的文本；⑧测试本地化后的软件，确保格式和内容正确。

要注意的是软件本地化不等于翻译，因为翻译的主要任务是把源语言转换到另一种目标语言，翻译是本地化的子集，当文字被翻译后，还要对产品进行许多相应的修改，包括技术层面的更改和文化层面的更改。

3. 软件全球化

软件全球化是一个概念化产品的过程，它基于全球市场考虑，为全球用户设计，面向全球市场发布具有一致界面、风格和功能的软件。它的核心特征和代码设计并不局限于一种语言和区域用户，可以支持不同目标市场的语言文字和数据信息的输入、输出、显示和存储。

4. 软件中文化

“中文化”是一个很模糊的概念。软件的“中文化”既包含使软件国际化，又包含使软件本

地化。也就是说,"中文化"不仅仅是只把软件本地化这么简单的事情,更重要的是因为目前操作系统直接支持中文的软件太少,做"中文化"必须先做"国际化"。

另外,由于历史的原因,现阶段使用的中文又有简体中文和繁体中文之分,所使用的编码也不同。支持中文的软件应该同时支持简体中文和繁体中文,这对软件的国际化提出了更高的要求。

12.1.2 常用字符集编码及 UTF-8

在软件"国际化"和"本地化"过程当中,最核心的问题就是字符集编码的使用问题。这个问题最直接和最直观的后果是软件运行中出现乱码现象。直接原因是字符在保存时的编码格式和要显示的编码格式不一样。因为,我们的软件系统,从底层数据库编码、应用程序编码到 GUI 界面编码,如果有一项不一致的话,就会出现乱码。所以,解决乱码问题关键是让交互系统之间编码一致。

1. 字符集编码概念

字符(Character)是文字与符号的总称,包括文字、图形符号、数学符号等。一组抽象字符的集合就是字符集(Charset)。

字符集常常和一种具体的语言文字对应起来,该文字中的所有字符或者大部分常用字符就构成了该文字的字符集,如英文字符集。另外,一组有共同特征的字符也可以组成字符集,比如繁体汉字字符集、日文汉字字符集。当然,字符集的子集也是字符集。

计算机要处理各种字符,就需要将字符和二进制内码对应起来,这种对应关系就是字符编码(Encoding)。制定编码首先要确定字符集,并将字符集内的字符排序,然后和二进制数字对应起来。根据字符集内字符的多少,会确定用几个字节来编码。每种编码都限定了一个明确的字符集合,叫做已编码字符集(Coded Character Set),这是字符集的另外一个含义。通常所说的字符集大多是这个含义。

2. 字符集种类

1) ASCII

ASCII(American Standard Code for Information Interchange,美国信息交换标准码),是目前计算机中用得最广泛的字符集及其编码,由美国国家标准局(ANSI)制定。它已被国际标准化组织(ISO)定为国际标准,称为 ISO 646 标准。

ASCII 字符集由控制字符和图形字符组成。在计算机的存储单元中,一个 ASCII 码值占一个字节(8 个二进制位),其最高位(b7)用做奇偶校验位。

所谓奇偶校验,是指在代码传送过程中用来检验是否出现错误的一种方法,一般分奇校验和偶校验两种:①奇校验规定——正确的代码一个字节中 1 的个数必须是奇数,若非奇数,则在最高位 b7 添 1;②偶校验规定——正确的代码一个字节中 1 的个数必须是偶数,若非偶数,则在最高位 b7 添 1。

2) ISO 8859-1

ISO 8859,全称 ISO/IEC 8859,是国际标准化组织(ISO)及国际电工委员会(IEC)联合制定的一系列 8 位字符集的标准,现时定义了 15 个字符集。

ASCII 收录了空格及 94 个"可印刷字符",足以给英语使用。

但是,其他使用拉丁字母的语言(主要是欧洲国家的语言),都有一定数量的变音字母,故可以使用 ASCII 及控制字符以外的区域来储存及表示。

除了使用拉丁字母的语言外,使用西里尔字母的东欧语言、希腊语、泰语、现代阿拉伯语、

希伯来语等，都可以使用这个形式来储存及表示。

ISO 8859-1 (Latin-1)——西欧语言。

ISO 8859-2 (Latin-2)——中欧语言。

ISO 8859-3 (Latin-3)——南欧语言。世界语也可用此字符集显示。

ISO 8859-4 (Latin-4)——北欧语言。

ISO 8859-5 (Cyrillic)——斯拉夫语言。

ISO 8859-6 (Arabic)——阿拉伯语。

ISO 8859-7 (Greek)——希腊语。

ISO 8859-8 (Hebrew)——希伯来语(视觉顺序)。

ISO 8859-8-I——希伯来语(逻辑顺序)。

ISO 8859-9 (Latin-5 或 Turkish)——它把 Latin-1 的冰岛语字母换走，加入土耳其语字母。

ISO 8859-10 (Latin-6 或 Nordic)——北日耳曼语支，用来代替 Latin-4。

ISO 8859-11 (Thai)——泰语，从泰国的 TIS620 标准字集演化而来。

ISO 8859-13 (Latin-7 或 Baltic Rim)——波罗的语族。

ISO 8859-14 (Latin-8 或 Celtic)——凯尔特语族。

ISO 8859-15 (Latin-9)——西欧语言，加入 Latin-1 欠缺的法语及芬兰语重音字母，以及欧元符号。

ISO 8859-16(Latin-10)——东南欧语言。主要供罗马尼亚语使用，并加入欧元符号。

很明显，ISO 8859-1 编码表示的字符范围很窄，无法表示中文字符。

但是，由于是单字节编码，和计算机最基础的表示单位一致，所以很多时候，仍旧使用 ISO 8859-1 编码来表示。而且在很多协议上，默认使用该编码。

3) UCS

通用字符集(Universal Character Set，UCS)是由 ISO 制定的 ISO 10646(或称 ISO/IEC 10646)标准所定义的字符编码方式，采用 4 字节编码。UCS 包含了已知语言的所有字符(除了拉丁语、希腊语、斯拉夫语、希伯来语、阿拉伯语、亚美尼亚语、格鲁吉亚语，还包括中文、日文、韩文这样的象形文字，以及大量的图形、印刷、数学、科学符号)。

UCS-2：与 Unicode 的 2byte 编码基本一样。

UCS-4：4byte 编码，目前是在 UCS-2 前加上 2 个全零的 byte。

4) Unicode

Unicode(统一码、万国码、单一码)是一种在计算机上使用的字符编码。它是 http://www.unicode.org 制定的编码机制，要将全世界常用文字都涵括进去。它为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。

1990 年开始研发，1994 年正式公布。随着计算机工作能力的增强，Unicode 也在面世以来的十多年里得到普及。

但自从 Unicode 2.0 开始，Unicode 采用了与 ISO 10646-1 相同的字库和字码，ISO 也承诺 ISO 10646 将不会给超出 0x10FFFF 的 UCS-4 编码赋值，使得两者保持一致。

Unicode 的编码方式与 ISO 10646 的通用字符集(Universal Character Set，UCS)概念相对应，目前常用的 Unicode 版本对应于 UCS-2，使用 16 位的编码空间。也就是每个字符占用 2 个字节，基本满足各种语言的使用。实际上目前版本的 Unicode 尚未填充满这 16 位编码，保留了大量空间作为特殊使用或将来扩展。

5）UTF

Unicode的实现方式不同于编码方式。一个字符的Unicode编码是确定的，但是在实际传输过程中，由于不同系统平台的设计不一定一致，以及出于节省空间的目的，对Unicode编码的实现方式有所不同。

Unicode的实现方式称为Unicode转换格式（Unicode Translation Format，UTF）。

UTF-8：8bit变长编码，对于大多数常用字符集（ASCII中0—127字符）它只使用单字节，而对其他常用字符（特别是朝鲜和汉语会意文字），它使用3字节。

UTF-16：16bit编码，是变长码，大致相当于20位编码，值在0到0x10FFFF之间，基本上就是Unicode编码的实现，与CPU字序有关。

6）汉字编码

目前涉及的汉字编码有：①GB2312字集是简体字集，全称为GB2312(80)字集，共包括国标简体汉字6763个；②BIG5字集是台湾繁体字集，共包括国标繁体汉字13 053个；③GBK字集是简繁字集，包括GB字集、BIG5字集和一些符号，共包括21 003个字符；④GB18030是国家制定的一个强制性大字集标准，全称为GB18030—2000，它的推出使汉字集有了一个“大一统”的标准。

GB2312是基于区位码设计的，区位码把编码表分为94个区，每个区对应94个位，每个字符的区号和位号组合起来就是该汉字的区位码。区位码一般用十进制数来表示，如1601就表示16区1位，对应的字符是“啊”。在区位码的区号和位号上分别加上0xA0就得到了GB2312编码。

GB2312的编码范围是0xA1A1-0xFEFE，去掉未定义的区域之后可以理解为实际编码范围是0xA1A1-0xF7FE。

区位码更应该认为是字符集的定义，定义了所收录的字符和字符位置，区位码和GB2312编码的关系有点像Unicode和UTF-8。

GBK编码是GB2312编码的超集，向下完全兼容GB2312，同时GBK收录了Unicode基本多文种平面中的所有CJK汉字。GBK的整体编码范围是0x8140-0xFEFE，不包括低字节是0×7F的组合。高字节范围是0×81-0xFE，低字节范围是0x40-7E和0x80-0xFE（低字节是0x40-0x7E的GBK字符有一定特殊性，因为这些字符占用了ASCII码的位置，这样会给一些系统带来麻烦）。

GB18030编码向下兼容GBK和GB2312，兼容的含义是不仅字符兼容，而且相同字符的编码也相同。GB18030收录了所有Unicode 3.1中的字符，包括中国少数民族字符，可以说是世界大多数民族的文字符号都被收录在内。

GBK和GB2312都是双字节等宽编码，如果再考虑与ASCII兼容的单字节，也可以将它们理解为是单字节和双字节混合的变长编码。GB18030编码是变长编码，有单字节、双字节和四字节三种方式。

GB18030的单字节编码范围是0x00-0x7F，完全等同于ASCII；双字节编码的范围和GBK相同，高字节是0x81-0xFE，低字节的编码范围是0x40-0x7E和0x80-FE；四字节编码中第一、三字节的编码范围是0x81-0xFE，二、四字节是0x30-0x39。

3. 编程语言与编码

C、C++、Python2内部字符串都是使用当前系统默认编码；Python3、Java内部字符串用Unicode保存；Ruby有一个内部变量$KCODE用来表示可识别的多字节字符串的编码，变量值为EUC、SJIS、UTF8、NONE之一。

4. 字符编码的选择

考虑到 UTF-8 是 Unicode 的一种可行的编码方式(即将字符映射为字节序列的方式),它本身仅仅只是为抽象字符实体分配相应的数值。可以使用 Unicode 的许多不同的编码方式,包括 UTF-8、UTF-16(及其相关的、过时的 UCS-2)和 UTF-32(也称为 UCS-4)。

与其他编码方式相比,UTF-8 具有一些优点:它兼容 ASCII,所以老的应用程序一般都可以处理 UTF-8 文本(尽管它们无法理解 127 以上的值)。另外,UTF-8 具有很高的效率(特别是对于使用西方语言的文本)、针对传输错误具有很高的健壮性(在出现损坏时,最多丢失一个额外的字符),并且它已得到了广泛的认可,越来越多的应用程序可以理解和处理它。

GBK 的文字编码是用双字节来表示的,即不论中、英文字符均使用双字节来表示,为了区分中文,将其最高位都设定成 1。GBK 包含全部中文字符,是国家编码,通用性比 UTF8 差,不过 UTF8 占用的数据库比 GBK 大。

GBK、GB2312 等与 UTF8 之间都必须通过 Unicode 编码才能相互转换。

由于上述这些原因,UTF-8 通常是正确的选择。当然,我们也有可能希望避免某些处理开销,或者甘愿牺牲内存,那么在这样的情况下,我们可以在内部使用 UCS-4。但是对于所有的外部通信,我们需要使用 UTF-8。

然而关于 UTF-8,有一点我们必须清楚:UTF-8 是一种多字节编码系统,这意味着在完成对下一个字符的解码工作之前,我们不可能知道它将占用多少个字节。所以,我们无法使用指针运算来遍历 UTF 字符。相反,要始终使用专用的、可以识别 UTF 的函数来完成这项工作。

12.2 软件本地化测试

软件本地化测试的测试对象就是本地化的软件,需要在本地化的操作系统上进行。虽然本地化的软件是基于源程序软件创建的,但是两者的测试内容和重点有很大的不同:①测试顺序不同。首先要先对源程序软件进行测试,然后再创建本地化软件,测试本地化软件。②测试内容和重点不同。源程序软件主要测试功能和性能,结合软件界面测试。本地化软件的测试,更注重因本地化引起的错误,例如翻译是否正确,本地化的界面是否美观,本地化后的功能是否与源语言软件保持一致。③测试环境不同。源程序软件测试通常在源语言的操作系统上进行。本地化软件在本地化的操作系统上进行。

12.2.1 本地化之前的国际化测试

软件国际化的测试就是验证软件产品是否支持多字节字符集、区域设置、时区设置、界面定制性、内嵌字符串编码和字符串扩展等。软件国际化的测试通常在本地化开始前进行,以识别潜在的不支持软件国际化特性的问题。

1. 国际化测试方法

理想的情况是,国际化测试在英文版本完成时就已结束。但实际上,设计评审和代码审查是国际化测试中最有效的方法,首先在设计上,要验证其是否遵守软件国际化的软件开发标准,是否具有国际化特性的一些基本功能——用户的时区、语言和地区等设置,然后审查程序代码和资源文件,确认源代码和显示内容是否被分离、是否使用各类正确的数据格式处理函数等。代码审查,可以采用走查的方法。先列出一个简单的检查列表(Checklist),依据这个列表,从头到尾快速地浏览所有代码,确保在代码上对 I18N 的充分支持。通过代码审查,可以发现大部分有关 I18N 的问题。

除了设计评审和代码审查之外，I18N 测试有两种基本方法：①针对源语言的功能测试。在源语言版本中，直接检查某些功能特性是否符合要求，如不同的区域设置、不同的时区显示等。②针对伪翻译（Pseudocode，Pseudo-translation）版本的测试。即文字、图片信息中的源语言被混合式的多种语言（如英文、中文、日文和德文等）替代，然后进行全面的 I18N 测试，包括相关的功能测试、界面测试，但不包括翻译验证等。

2. 伪翻译测试

伪翻译（Pseudo Translation）是软件国际化测试的重要手段之一，它可以选择一种以上的本地化语言模拟本地化处理的结果。可以在进行实际本地化处理之前预览和查看本地化的问题。通过伪本地化翻译，可以发现源语言软件的国际化设计中的错误，方便后续本地化时处理，提高软件的可本地化能力。在某些本地化工具中，可以设置在进行了本地化翻译后字符长度的扩展比例、替换字符、前缀和后缀字符。

1）针对源语言的功能测试

I18N 的测试不同于本地化的测试，其中部分测试工作可以在源语言中进行。假定源语言是英文，我们可以在英文版本中进行下列测试：①时区的设置及其相应的时间显示；②地区的设置及其相应的日期、货币显示；③可以选择不同的语言；④输入多字节字符串。

2）针对伪翻译版本的测试

源语言的测试具有局限性，不能完全验证软件产品是否能很好地支持多字节字符集，例如弹出的窗口是否可以根据显示内容的长度进行自我调整？窗口中显示的内容是否会出现乱码现象等？这时，可采用伪翻译（Pseudo-translation 或 Pseudocode）的版本，进一步完成 I18N 的测试。采用伪翻译版本，可以完成这些测试任务：①测试多字节字符集和脚本；②测试多字节字符串的输入显示是否正确；③测试多字节字符文件、文件名、文件夹及其处理；④测试索引和排序；⑤测试本地化的操作系统、键盘支持等。

如果采用正式翻译的版本，也可以进行相关的 I18N 测试，例如当源语言是英文，可以采用繁体中文、日文和阿拉伯文等作为几种典型的语言版本来对 I18N 进行更充分的测试。I18N 测试的重点在功能上，但将 2～3 种语言的本地化测试和 I18N 测试结合起来也没有坏处。采用伪翻译版本的好处是：①可更早地进行 I18N 测试，能够比较快也更容易在源语言版本之上构造伪翻译版本，因为不需要准确翻译，甚至不需要翻译，仅仅是替换成不同语言的文字；②可以一举两得，即同时验证多种语言的不同特性，如中文的多字节文字精练简短，德文的长度，以及中英文混合等；③给测试者一个清晰的信号，让他知道这是 I18N 测试，不是 L10N 测试。

12.2.2　软件本地化测试方法

1. 软件本地化测试的目的

软件本地化测试的目的是发现和报告本地化软件的错误和缺陷。通过对这些错误和缺陷的处理，确保本地化软件的语言质量、互操作性、功能等符合软件本地化的设计要求，满足当地语言市场用户的使用要求。通过分析错误产生的原因和错误的分布特征，可以帮助项目管理者发现当前所采用的软件测试过程的缺陷，以便改进。同时这种分析也能帮助设计出有针对性的检测方法，改善测试的有效性。

软件本地化测试是对本地化软件质量控制的重要手段，是运行本地化软件程序寻找和发现错误的质量控制过程。更详细的定义可以描述为，软件本地化测试是根据软件本地化各阶段的测试计划和规格说明，精心设计一批测试用例（即输入数据及其输出结果），并利用这些测

试用例去运行本地化软件，以发现程序错误和缺陷的过程。软件本地化测试的目标是以最少的时间、人力和软硬件资源，找出本地化软件中的各种类型的错误和缺陷。测试应该能验证本地化软件的功能和性能与源语言软件保持一致，本地化软件的语言质量、软件界面、文档内容等符合当地语言市场用户的使用要求，符合特定区域的文化传统和风俗习惯。

2. 软件本地化测试的特点

软件本地化测试除了具有一般软件测试的特征外，还有其特有的特征。

(1) 本地化测试对语言的要求较高。不仅要准确理解英文(测试的全部文档，例如测试计划、测试用例、测试管理文档、工作邮件都是英文的)，还要精通本地语言。例如测试简体中文的本地化产品，我们完全胜任；而测试德语本地化软件，则需要母语是德语的测试人员。

(2) 本地化测试以手工测试为主，但是经常使用许多定制的专用测试程序。手工测试是本地化测试的主要方法，但为了提高效率，满足特定测试需要，经常使用各种专门开发的测试工具。一般这些测试工具都是由开发英文软件的公司的开发人员或测试开发人员开发的。

(3) 本地化测试通常采用外包测试进行。为了降低成本，保证测试质量，国外大的软件开发公司都把本地化的产品外包给各个不同的专业本地化服务公司，软件公司负责提供测试技术指导和测试进度管理。

(4) 本地化测试的缺陷具有规律性特征。本地化缺陷主要包括语言质量缺陷、用户界面布局缺陷、本地化功能缺陷等，这些缺陷具有比较明显的特征，采用规范的测试流程，可以发现绝大多数缺陷。

(5) 本地化测试特别强调交流和沟通。由于实行外包测试，本地化测试公司要经常与位于国外的软件开发公司进行有效交流，以便使测试按照计划和质量完成。有些项目需要每天与客户交流，发送进度报告。更多的是每周报告进度，进行电话会议、电子邮件等交流。此外，本地化测试公司内部的测试团队成员也经常交流彼此的进度和问题。

3. 软件本地化错误类型及其原因

软件本地化的错误主要分为两大类：一类是由于源程序软件编码错误引起的，另一类是由软件本地化引起的。综合分析本地化软件的错误类型，可以分为 4 种：翻译错误、功能错误、国际化错误和本地化错误。

1) 翻译错误

翻译错误是指在软件本地化中由于翻译不当而引起的错误。这类错误产生的原因：①翻译人员不熟悉翻译要求；②翻译人员的工作疏漏；③用户界面的翻译与标准词汇表不一致。

主要表现有：①应该翻译而没有翻译的英文字符；②不应该翻译而翻译的本地化字词；③错误翻译的字词；④只在本地化版本中存在该类型错误；⑤较多隐含在对话框各控件以及帮助文档中。

2) 功能错误

功能错误即软件中的某些功能无效。这类错误产生原因有：①软件开发编程错误，引起的某些功能错误，这些功能错误在源语言软件和本地化软件中都存在；②由于本地化过程产生的某些功能错误，这些错误仅出现在本地化软件中。

错误特点：①经常出现在软件的菜单项、工具栏按钮和对话框的功能按钮中。②不能实现设计要求的功能。绝大多数存在于源语言软件和本地化软件中，也有的仅出现在本地化软件中。③产生与设计要求不符合的结果。

3) 国际化错误

国际化错误即源语言软件在开发中没有正确地进行国际化设计而导致的错误。产生原因

有：①源程序在设计时没有正确进行国际化设计，例如，没有提供双字节字符集的支持；②单字节字符向双字节字符转化过程中，单字节和双字节之间的差别，可能使得某些本地化后的双字节字符的显示乱码。

错误特点：①出现在本地化后的版本中；②不支持双字节字符的输入和输出，包括双字节的文件名和路径名；③控件或对话框中显示不可辨识或无意义的明显错误的字符；④本地化软件的列表项排序错误；⑤某些没有本地化的字符串；⑥与当地不符合的日期和时间格式。

4）本地化错误

本地化错误即本地化过程中引起的错误，与源语言功能不一致，本地化错误只存在于本地化软件中。这类错误是我们软件本地化测试的重点。产生原因有：①软件本地化后，由于源语言和本地化语言的表达方式不同，本地化后的字符数与源语言不同，每个字符所占空间尺寸不同，使得在英文版本中正确显示的控件字符，可能在本地化版本中显示不正确。②在编译本地化软件之前，没有对资源文件对话框及其控件调整大小。③本地化人员调整软件资源文件不当引起。例如，对话框及其控件高度或宽度的不正确调整。

错误类型及特点：①只在本地化版本中存在该类型错误；②热键冲突，热键丢失，热键无效，热键错误；③应该翻译而没有翻译，不该翻译却翻译了；④控件中字符显示不完整，文字越界，控件相互重叠或排列不均匀，出现垃圾字符；⑤应该显示的页面不能正常显示，链接出错，丢失行，丢失菜单项，较多隐含在本地化版本的对话框各控件及帮助文档中。

每种类型的错误的数量不同，这与源程序软件和本地化软件的质量有密切关系。如果源程序软件没有经过完整的测试，包括功能测试和本地化性能测试，那么本地化软件中就将存在很多功能错误、界面错误、双字节错误。如果本地化软件没有经过良好的本地化处理，将会产生很多翻译错误和界面错误。

4. 软件本地化测试的类型

软件本地化测试是在本地化的操作系统上对本地化的软件版本进行测试。根据软件本地化项目的规模、测试阶段以及测试方法，本地化测试分为多种类型，每种类型都对软件本地化的质量进行了检测和保证。为了提高测试的质量，保证测试的效率，不同类型的本地化测试需要使用不同的方法，掌握必要的测试技巧。我们在这里主要对本地化测试中具有代表性的测试类型进行介绍。

1）导航测试

导航测试（Pilot Testing）是为了降低软件本地化的风险而进行的一种本地化测试。大型的全球化软件在完成国际化设计后，通常选择少量的典型语言进行软件的本地化，以此测试软件的可本地化能力，降低多种语言同时本地化的风险。

导航测试是用于实现对数种语言进行本地化而新开发软件的一种很重要的测试，导航测试的语言主要由语言市场的重要性和规模确定，也要考虑语言编码等的代表性。例如，德语市场是欧洲的重要市场，通常作为导航测试的首要单字节字符集语言。日语是亚洲重要的市场，可以作为双字节字符集语言代表。随着中国国内软件市场规模的增加，国际软件开发商逐渐对简体中文本地化提高重视程度，简体中文有望成为更多导航测试的首选语言。

导航测试是软件本地化项目早期进行的探索性测试，需要在本地化操作系统上进行，测试的重点是软件的国际化能力和可本地化能力，包括与区域相关的特性的处理能力，也包括测试是否可以容易地进行本地化，减少硬编码等缺陷。由于导航测试在整个软件本地化过程中意义重大，而且导航测试的持续时间通常较短，另外由于是新开发的软件的本地化测试，测试人

员对软件的功能和使用操作了解不多，因此，本地化公司通常需要在正式测试之前收集和学习软件的相关资料，做好测试环境和人员的配备，配置具有丰富测试经验的工程师执行测试。

2）可接受性测试

本地化软件的可接受性测试（Build Acceptable Testing）也称做冒烟测试（Smoke Testing），是指对编译的软件本地化版本的主要特征进行基本测试，从而确定版本是否满足详细测试的条件。理论上，每个编译的本地化新版本在进行详细测试之前，都需要进行可接受性测试，以便在早期发现软件版本的可测试性，避免不必要的时间浪费。

注意，软件本地化版本的可接受性测试与软件公司为特定客户定制开发的原始语言软件在交付客户前的验收测试完全不同。验收测试主要确定软件的功能和性能是否达到了客户的需求，如果一切顺利，只进行一次验收测试就可以结束。

本地化软件在编译后，编译工程师通常需要执行版本健全性检查（Build Sanity Check），确定本地化版本的内容和主要功能可以用于测试。而编译的本地化版本是否真的满足测试条件则还要通过独立的测试人员进行可接受性测试，它要求测试人员在较短的时间内完成，确定本地化的软件版本是否满足全面测试的要求，是否正确包含了应该本地化的部分。如果版本通过了可接受性测试，则可以进入软件全面详细测试阶段；反之，则需要重新编译本地化软件版本，直到通过可接受性测试。

在进行本地化软件版本的可接受性测试时，需要配置正确的测试环境（软件和硬件）。

在本地化的操作系统上安装软件，确定是否可以正确安装；运行软件，确定软件包含了应该本地化的全部内容，并且主要功能正确；然后卸载软件，保证软件可以彻底卸载。软件的完整性是需要注意的一个方面，通过使用文件和文件夹的比较工具软件，对比安装后的本地化软件和英文软件内容的异同，确定本地化的完整性。

3）语言质量测试

语言质量测试是软件本地化测试的重要组成部分，贯穿于本地化项目的各个阶段。语言质量测试的主要内容是软件界面和联机帮助等文档的翻译质量，包括正确性、完整性、专业性和一致性。

为了保证语言测试的质量，应该安排本地化语言是母语的软件测试工程师进行测试，同时请本地化翻译工程师提供必要的帮助。在测试之前，必须阅读和熟悉软件开发商提供的软件术语表（Glossary），了解软件翻译风格（Translation Style）的语言表达要求。

由于软件的用户界面总是首先进行本地化，因此，本地化测试初期的软件版本的语言质量测试主要以用户界面的语言质量为主，重点测试是否存在未翻译的内容，翻译的内容是否正确，是否符合软件术语表和翻译风格要求，是否符合母语表达方式，是否符合专业和行业的习惯用法。

本地化项目后期要对联机帮助和相关文档（各种用户使用手册等）进行本地化，这个阶段的语言质量测试，除了对翻译的表达正确性和专业性进行测试之外，还要注意联机帮助文件和软件用户界面的一致性。如果对于某些软件专业术语的翻译存在疑问，需要报告一个翻译问题，请软件开发商审阅，如果确认是翻译错误，需要修改术语表和软件的翻译。

关于本地化软件的语言质量测试，一个值得注意的问题是“过翻译”，就是软件中不应该翻译的内容（如软件的名称等）如果进行了翻译，应该报告软件“过翻译”错误。

4）用户界面测试

本地化软件的用户界面测试（UI Testing）也称做外观测试（Cosmetic Testing），主要对软件的界面文字和控件布局（大小和位置）进行测试。用户界面至少包括软件的安装和卸载界

面、软件的运行界面和软件的联机帮助界面。软件界面的主要组成元素包括窗口、对话框、菜单、工具栏、状态栏、屏幕提示文字等内容。

用户界面的布局测试是本地化界面测试的重要内容。由于本地化的文字通常比原始开发语言长度长，所以一类常见的本地化错误是软件界面上的文字显示不完整，例如按钮文字只显示一部分；另一类常见的界面错误是对话框中的控件位置排列不整齐，大小不一致。

相对于其他类型的本地化测试，用户界面测试可能是最简单的测试类型，软件测试工程师不需要过多的语言翻译知识和测试工具。但是由于软件的界面众多，而且某些对话框可能隐藏比较深入，因此，软件测试工程师必须尽可能地熟悉被测试软件的使用方法，这样才能找出那些较为隐蔽的界面错误。另外，某些界面错误可能是一类错误，需要报告一个综合的错误。例如，软件安装界面的“上一步”或“下一步”按钮显示不完整，则可能所有安装对话框的同类按钮都存在相同的错误。

5）功能测试

原始语言开发的软件的功能测试主要测试软件的各项功能是否实现以及是否正确，而本地化软件的功能测试主要测试软件经过本地化后，软件的功能是否与源软件一致，是否存在因软件本地化而产生的功能错误，例如，某些功能失效或功能错误。

本地化软件的功能测试相对于其他测试类型具有较大难度，由于大型软件的功能众多，而且有些功能不经常使用，可能需要多步组合操作才能完成，因此本地化软件的功能测试需要测试工程师熟悉软件的使用操作，对于容易产生本地化错误之处能够预测，以便减少软件测试的工作量，这就要求测试工程师具有丰富的本地化测试经验。

除了某些菜单和按钮的本地化功能失效错误外，本地化软件的功能错误还包括软件的热键和快捷键错误，例如，菜单和按钮的热键与源软件不一致或者丢失热键。另外一类是排序错误，例如，排序的结果不符合本地化语言的习惯。

发现本地化功能错误后，需要在源软件上进行相同的测试，如果源软件也存在相同的错误，则不属于本地化功能错误，而属于源软件的设计错误，需要报告源软件的功能错误。

另外，如果同时进行多种本地化语言（例如简体中文、繁体中文、日文和韩文）的测试，在一种语言上的功能错误也需要在其他语言版本上进行相同的测试，以确定该错误是单一语言特有的，还是许多本地化版本共有的。

软件的测试类型数量众多，可谓五花八门，而软件本地化测试又具有其自身的特点，除以上常见的本地化测试类型外，还包括联机帮助测试、本地化能力测试等。不论何种类型的本地化测试，其最终测试目标都是尽早找出软件本地化错误，保证本地化软件与原始开发语言软件具有相同的功能。通过正确配置本地化测试环境，合理组织本地化测试人员，采用正确的本地化流程和测试工具，完善软件缺陷的报告和跟踪处理，来保证软件本地化测试的有效实现。

5. 软件本地化测试的原则

测试原则规定了测试过程中应该遵循的基本思路，软件本地化测试的原则如下。

（1）在本地化软硬件环境中测试本地化软件。为了尽量符合本地化软件的使用环境和习惯，应该在本地化的操作系统上安装和测试本地化软件，使用当地语言市场的通用硬件及当地布局的键盘等，这样可以发现更多的本地化软件的区域语言、操作系统和硬件的兼容性问题。为了便于参考和对比，必须将源语言（例如英语）软件安装在源语言操作系统上。

（2）尽早地和不断地进行软件测试。软件本地化测试不是软件本地化的一个独立阶段，它贯穿于软件本地化项目的各个阶段。测试计划、测试用例等测试要素要在测试本地化软件版本（Build）前准备好。一旦得到可以测试的软件本地化版本，就立刻组织测试。争取尽早发

现更多的错误,把出现的错误在早期进行修复处理,减少后期修复错误时耗费过多的时间和人力。软件本地化测试工作强调的是发现软件因本地化产生的错误。不要过多地耗费时间测试软件的功能,因为本地化测试前,源语言软件已经进行过功能测试和国际化测试。所以,应该将本地化测试的重点放在本地化方面的错误,例如语言表达质量,软件界面布局,本地化字符的输入、输出和显示等。

(3) 软件错误报告、软件错误修复和软件错误修复验证应该由不同的软件工程师处理。为了保证软件测试效果,软件错误报告应该由测试工程师负责,软件错误修复应该由负责错误确认和处理的软件工程师负责,软件错误修复后的验证和关闭应该由软件错误报告者(测试工程师)负责。

习题

1. 什么是软件国际化?什么是软件本地化?如何使软件国际化和本地化?
2. 简述字符集编码概念,常用的字符集编码有哪些?
3. 软件国际化测试内容有哪些?如何进行软件国际化测试?
4. 简述软件本地化测试概念,软件本地化测试的主要目的是什么?
5. 软件本地化测试所具有的特点有哪些,如何进行测试?
6. 软件本地化测试包括哪些内容,为什么要对它们进行测试?



第13章 面向对象软件测试

面向对象技术是现在很流行的软件开发技术，正逐渐代替之前被广泛使用的面向过程开发方法，被认为是解决软件危机的新兴技术。面向对象技术产生更好的系统结构、更规范的编程风格，极大地优化了数据使用的安全性，提高了程序代码的重用，一些人就此认为随着面向对象分析 OOA 和面向对象设计 OOD 的成熟，更多的设计模式重用将减轻面向对象系统的繁重测试量。应该看到，尽管面向对象技术的基本思想是要保证软件质量，但实际情况并非如此。因为无论采用什么样的编程技术，编程人员的错误都是不可避免的，而且由于用面向对象技术开发的软件代码重用率高，因而就更需要严格测试，避免错误的繁衍。被重用代码在每次一个新的使用环境，都要谨慎地重新测试。为了获得面向对象系统的高可靠性，可能将需要更多，而不是更少的测试。因此，软件测试并没有因面向对象编程的兴起而丧失它的重要性。

传统的软件测试策略是从“小型测试”开始，逐步走向“大型测试”。即从单元测试开始，然后逐步进入集成测试，最后是有效性和系统测试。在传统的测试中，单元测试集中在最小的可编译程序单位——子程序(如模块、子程序、进程)，一旦这些单元均被独立测试后，它们就被集成在程序结构中，这时要进行一系列的集成测试，以发现由于模块接口所带来的错误和新单元加入所导致的负面作用。最后，系统被作为一个整体进行测试以保证发现在需求中的错误。

面向对象技术所独有的多态、继承、封装等新特性，导致了传统程序设计不会产生的错误出现的可能性。一度实践证明行之有效的软件测试对面向对象技术开发的软件多少显得有些力不从心。例如，在传统的面向过程程序中，对于函数

```
y = Function(x);
```

你只需要考虑一个函数(Function())的行为特点，而在面向对象程序中，我们不得不同时考虑基类函数(Base::Function())的行为和继承类函数(Derived::Function())的行为。

面向对象程序的结构不再是传统的功能模块结构，由于面向对象软件的封装性导致其没有传统结构化程序的层次式控制结构，而是作为一个整体，因而原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已成为不可能。而且，面向对象软件抛弃了传统的开发模式，对每个开发阶段都有不同以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。单元测试失去了本身的多数意义，传统的自顶向下和自底向上的集成测试策略也有了较大的改变。

因此，传统的测试模型对面向对象软件已经不再适用。针对面向对象软件的开发特点，应该有一种新的测试模型。

13.1 面向对象程序设计语言对软件测试的影响

面向对象程序设计语言的出现不仅改变了程序设计的风格，而且影响了软件开发的全过程。面向对象的软件需求分析方法、设计方法也应运而生。然而，面向对象方法对软件测试的

影响直至近年来才开始为人们所注意。下面,我们首先分析面向对象程序设计语言的特征对软件测试的影响。

13.1.1 信息隐蔽对测试的影响

类的重要作用之一是信息隐蔽。它对类中所封装的信息的连接进行控制,从而避免类中有关实现细节的信息被错误地使用。而这样的细节性信息正是软件测试所不可忽略的。由于面向对象的软件系统在运行时刻由一组协调工作的对象组成,对象具有一定的状态,所以对于面向对象的程序测试来说,对象的状态是必须考虑的因素,测试应涉及对象的初态、输入参数、输出参数、对象的终态。对象的行为是被动的,它只有在接收有关信息后才被激活来进行所请求的工作,并将结果返回给发信者。在工作过程中对象的状态可能被修改,产生新的状态,面向对象软件测试的基本工作就是创建对象(包括初始化),面向对象发送一系列信息然后检查结果对象的状态,看其是否处于正确的状态。问题是对象的状态往往是隐蔽的,若类中未提供足够的存取函数来表明对象的实现方式和内部状态,则测试者必须增添这样的函数。因此,类的信息隐蔽机制给测试带来困难。

13.1.2 封装和继承对测试的影响

在面向对象的程序中,由于继承的作用,一个函数可能被封装在多个类中,子类中还可以对继承的特征进行重定义。问题是,未重定义的继承特征是否还需要进行测试呢?重定义的特征需要重新测试是显然的,但如何测试重定义的特征呢?E. J. Weyuker 曾经提出了 11 条基于程序的测试数据集的充分性公理,D. E. Perry 与 G. E. Kaiser 根据 Weyuker 公理对这些问题进行了讨论,结论是:封装和继承并未简化测试问题,反而使测试更加复杂。

13.1.3 集成测试

对于用传统程序设计语言编写的软件,软件测试人员普遍接受三个级别的测试:单元测试、集成测试和系统测试。一个单元可以是一个函数或一个模块,虽然单元的定义可能不同,但单元测试的目的都是通过测试残桩(stub)和驱动程序来模拟和该单元相关的其他单元,以验证该单元自身是否正确地工作。当各单元被分别测试后,集成测试根据设计阶段形成的功能分解树,自顶向下或自底向上逐步用各个单元来替换残桩和驱动程序。集成测试强调软件的结构和接口,测试者往往要关注模块间的组装和调用关系,模块间的接口方面的问题。而系统测试强调软件的行为,系统测试者从用户的角度,观察系统级的输入和输出,以确定系统的行为是否与需求定义一致。无论在哪个级别上进行测试,其测试过程均为输入测试数据、处理、验证输出结果这三个步骤。

对面向对象的程序测试应当分为多少级别尚未达成共识。一般认为,面向对象的程序也和其他语言的程序一样,都要进行系统级测试。M. D. Smith 和 D. J. Robson 从面向对象程序的结构出发,认为面向对象的程序测试应当分为 4 个级别:①方法级,考察一个方法对数据进行的操作;②类级,考察封装在一个类中的方法和数据之间的相互作用;③簇级,考察一组协同操作的类之间的相互作用;④系统级,考察由所有类和主程序构成的整个系统。

他们认为,上述 4 个级别中的方法级测试和传统的单元测试相对应。也有人认为在面向对象程序中,最小的可测试单元已不是方法,而是类和类的实例。应该说,这些看法和传统的单元测试中对于单元的认识还是一致的。已经有经验表明类级测试是必需的,也是发现错误的重要手段,以簇作为测试的最小单元会导致某些应当在开发周期中较早发现的错误延至系

统测试时才发现。但传统的集成测试和面向对象的程序测试中的对应尚没有一致的看法。显然,基于功能分解的自顶向下和自底向上的集成测试策略并不适用于面向对象方法构造的软件。在各个方法分别测试之后,每次任选一个方法集成到类中逐步进行测试直至形成一个完整的类的集成策略也未必合适,原因是各个方法之间可能有相互作用,某一方法可能要求对象处于某个特定的状态,而该状态必须由其他方法设置,所以还需要考虑集成的次序问题。

基于结构的传统集成策略并不适于面向对象的程序。这是因为面向对象的程序的执行实际上是执行一个由信息连接起来的方法序列,而这个方法序列往往是由外部事件驱动的。根据这一执行特性,P. C. Jorgensen 和 C. Erickson 认为,面向对象的测试应该分为 5 个层次:①方法测试;②方法/信息路径(MMPath)测试,即执行一个方法/信息序列直至到达一个不再发送信息的方法;③系统基本功能测试,即测试从一个输入端口事件开始,由此触发一个方法/信息路径的执行,最后终结于某一输出端口事件;④线程测试;⑤线程间相互作用测试。其中方法测试对应于单元测试,方法/信息路径测试和系统基本功能测试对应于集成测试,而线程测试和线程间相互作用测试对应于系统测试。

13.1.4　多态性和动态绑定对测试的影响

多态性和动态绑定为程序的执行带来了不确定性,给软件测试带来了新的挑战。多态性和动态绑定是面向对象方法的关键特性之一。同一消息可以根据发送消息对象的不同采用多种不同的行为方式,这就是多态的概念。如根据当前指针引用的对象类型来决定使用正确的方法,这就是多态性的行为操作。运行时系统能自动为给定消息选择合适的实现代码,这给程序员提供了高度柔性、问题抽象和易于维护。

例如:假设有类 A、类 B 和类 C 三个类,类 B 继承类 A,类 C 又继承类 B。成员函数 a()分别存在于这三个类中,但在每个类中的具体实现则不同。同时在程序中存在一个函数 fn(),该函数在形参中创建了一个类 A 的实例 Ca,并在函数中调用了方法 a()。程序运行时相当于执行了一个分情况语句 switch,首先判定传递过来的实参的类型(类 A 或类 B 或类 C),然后再确定究竟执行哪一个类中的方法 a()。

在测试时必须为每一个分支生成测试用例,以覆盖所有的分支和所有的程序代码。因而,多态性和动态绑定所带来的不确定性,使得传统测试实践中的静态分析方法遇到了不可逾越的障碍,也增加了系统运行中可能的执行路径,加大了测试用例选取的难度和数量。多态性给软件测试带来的问题仍然是目前研究的重点及难点问题之一。

从上述分析可知,虽然信息隐蔽与封装使得类具有较好的相对独立性,有利于提高软件的易测试性和保证软件的质量,但是,这些机制与继承机制和动态绑定给软件测试带来了新的难题。尤其是面向对象软件中类与类之间的集成测试和类中各个方法之间的集成测试具有特别重要的意义,与传统语言编写的软件相比,集成测试的方法和策略也应该有所不同。

13.2　面向对象测试模型

面向对象软件的开发是从分析和设计模型的创建开始的。模型从对系统需求相对非正式的表示开始,逐步演化为详细的类模型、类连接和关系、系统设计和分配,以及对象设计在每个阶段的测试模型。面向对象分析的测试包括对认定的对象的测试、对认定的结构的测试、对认定的主题的测试、对定义的属性和实例关联的测试、对定义的服务和消息关联的测试等内容。面向对象设计的测试包括对认定的类的测试、对构造的类层次结构的测试、对类库支持的测试

等内容。面向对象编程的测试主要是类测试，主要测试数据成员是否满足数据封装的要求和类是否实现了要求的功能。

面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析(OOA)、面向对象设计(OOD)和面向对象编程(OOP)三个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。由于面向对象的特点，采用这种开发模型能有效地将分析设计的文本或图表代码化，不断适应用户需求的变动。针对这种开发模型，结合传统的测试步骤的划分，这里建议一种整个软件开发过程中不断测试的测试模型，使开发阶段的测试与编码完成后的单元测试、集成测试、系统测试成为一个整体(参见表 13-1)。测试模型如图 13-1 所示。

表 13-1　面向对象开发与测试过程

OO：面向对象	OOA：面向对象分析
OOD：面向对象设计	OOP：面向对象编程
OOA Test：面向对象分析的测试	OOD Test：面向对象设计的测试
OOP Text：面向对象编程的测试	OO Unit Test：面向对象单元测试
OO Integrate Test：面向对象集成测试	OO System Test：面向对象系统测试

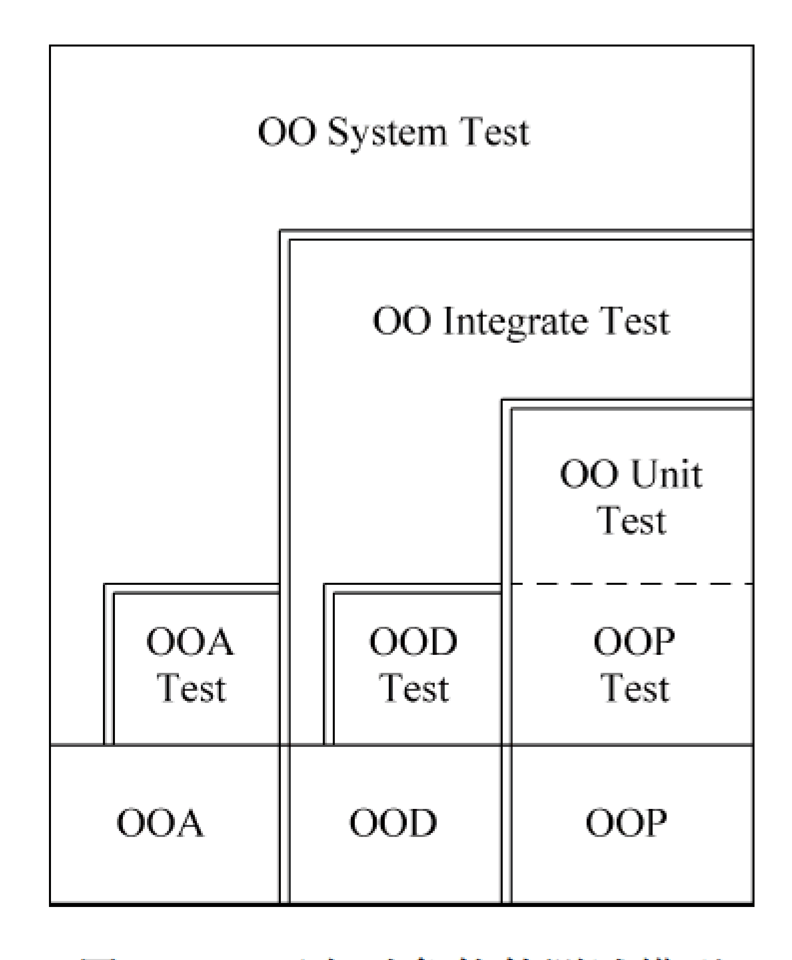


图 13-1　面向对象软件测试模型

OOA Test 和 OOD Test 是对分析结果和设计结果的测试，主要是对分析设计产生的文本进行，是软件开发前期的关键性测试。OOP Test 主要针对编程风格和程序代码实现进行测试，其主要的测试内容在面向对象单元测试和面向对象集成测试中体现。面向对象单元测试是对程序内部具体单一的功能模块的测试，如果程序是用 C++语言实现，那么主要就是对类成员函数的测试。面向对象单元测试是进行面向对象集成测试的基础。面向对象集成测试主要对系统内部的相互服务进行测试，如成员函数间的相互作用、类间的消息传递等。面向对象集成测试不但要基于面向对象单元测试，更要参见 OOD 或 OOD Test 结果(详见后面内容)。面向对象系统测试是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准，需要借鉴 OOA 或 OOA Test 结果。

尽管上述各阶段的测试构成了一相互作用的整体，但其测试的主体、方向和方法各有不同。

13.2.1　面向对象分析的测试

传统的面向过程分析是一个功能分解的过程，是把一个系统看成可以分解的功能的集合。这种传统的功能分解分析法的着眼点在于一个系统需要什么样的信息处理方法和过程，以过程的抽象来对待系统的需要。而面向对象分析（OOA）是把 E-R 图和语义网络模型，即信息造型中的概念，与面向对象程序设计语言中的重要概念结合在一起而形成的分析方法，最后通常是得到问题空间的图表的形式描述。

OOA 直接映射问题空间，全面地将问题空间中的解进行现实抽象化。将问题空间中的实例抽象为对象（不同于 C++中的对象概念），用对象的结构反映问题空间的复杂实例和复杂关系，用属性和服务表示实例的特性和行为。对于一个系统而言，与传统分析方法产生的结果相反，行为是相对稳定的，结构是相对不稳定的，这更充分反映了现实的特性。OOA 的结果是为后面阶段类的选定和实现、类层次结构的组织和实现提供平台。因此，OOA 对问题空间分析抽象得不完整，最终会影响软件的功能实现，导致软件开发后期大量可避免的修补工作；而一些冗余的对象或结构会影响类的选定、程序的整体结构或增加程序员不必要的工作量。因此，我们对 OOA 的测试重点在于其完整性和冗余性。

尽管 OOA 的测试是一个不可分割的系统过程，但是为了叙述方便，我们将 OOA 阶段的测试划分为 5 个方面：①对认定的对象的测试；②对认定的结构的测试；③对认定的主题的测试；④对定义的属性和实例关联的测试；⑤对定义的服务和消息关联的测试。

对象、结构、主题等在 OOA 结果中的位置，可以参考图 13-2。

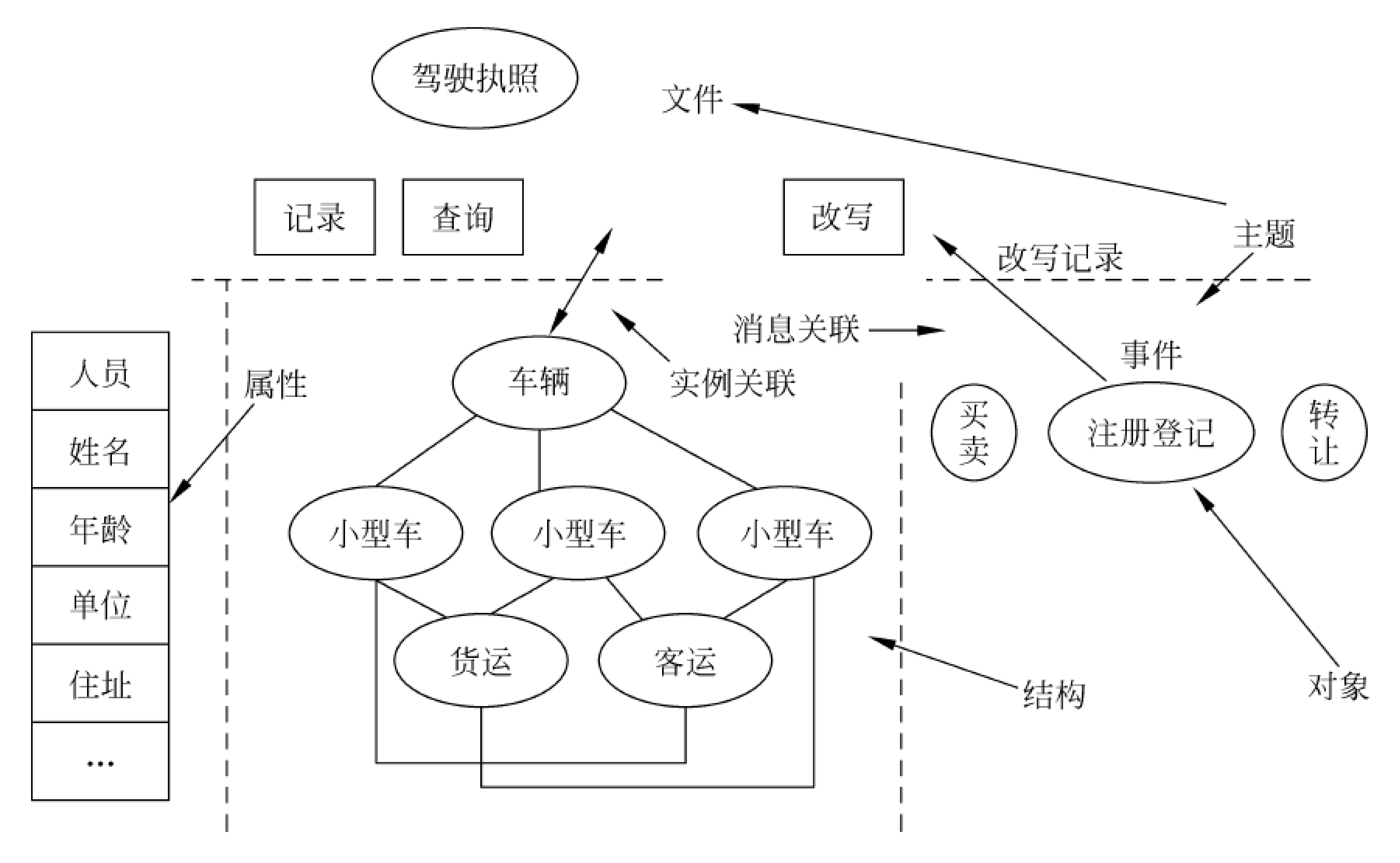


图 13-2　车辆管理系统部分 OOA 分析结构示意

1. 对认定的对象的测试

OOA 中认定的对象是对问题空间中的结构、其他系统、设备、被记忆的事件、系统设计的人员等实际实例的抽象。对它的测试可以从以下方面考虑：①认定的对象是否全面，是否问题空间中所有涉及的实例都反映在认定的抽象对象中；②认定的对象是否具有多个属性（只有一个属性的对象通常应看成其他对象的属性，而不是抽象为独立的对象）；③被认定为同一对象的实例是否有共同的区别于其他实例的属性；④对认定为同一对象的实例是否提供或需要相同的服务，如果服务随着不同的实例而变化，则认定的对象就需要分解或利用继承性来分

类表示；⑤如果系统没有必要始终保持对象代表的实例的信息，提供或者得到关于它的服务，则认定的对象也没必要去关注；⑥认定的对象的名称应该尽量准确、适用。

2. 对认定的结构的测试

认定的结构指的是多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系。认定的结构分为两种：分类结构和组装结构。分类结构体现了问题空间中实例的一般与特殊的关系，组装结构体现了问题空间中实例整体与局部的关系。

(1) 对认定的分类结构的测试可从以下几方面着手：①对于结构中的一种对象，尤其是处于高层的对象，是否在问题空间中含有不同于下一层对象的特殊可能性，即是否能派生出下一层对象；②对于结构中的一种对象，尤其是处于同一低层的对象，是否能抽象出在现实中有意义的更一般的上层对象；③对于所有认定的对象，是否能在问题空间内向上层抽象出在现实中有意义的对象；④高层的对象的特性是否完全体现下层的共性；⑤低层的对象是否有高层特性基础上的特殊性。

(2) 对认定的组装结构的测试可从以下几方面入手：①整体(对象)和部件(对象)的组装关系是否符合现实的关系；②整体(对象)的部件(对象)是否在考虑的问题空间中有实际应用；③整体(对象)中是否遗漏了反映在问题空间中有用的部件(对象)；④部件(对象)是否能够在问题空间中组装新的有现实意义的整体(对象)。

3. 对认定的主题的测试

主题是在对象和结构的基础上更高一层的抽象，是为了提供 OOA 分析结构的可见性，如同文章对各部分内容的概要。对主题层的测试应该考虑以下几方面：①贯彻 George Miller 的"7+2"原则，如果主题个数超过 7 个，就要求对有较密切属性和服务的主题进行归并；②主题所反映的一组对象和结构是否具有相同和相近的属性和服务；③认定的主题是否是对象和结构更高层的抽象，是否便于理解 OOA 结果的概貌(尤其是对非技术人员的 OOA 结果读者)；④主题间的消息联系(抽象)是否代表了主题所反映的对象和结构之间的所有关联。

4. 对定义的属性和实例关联的测试

属性是用来描述对象或结构所反映的实例的特性。而实例关联是用来反映实例集合间的映射关系。对属性和实例关联的测试可从以下几方面考虑：①定义的属性是否对相应的对象和分类结构的每个现实实例都适用；②定义的属性在现实世界是否与这种实例关系密切；③定义的属性是否能够不依赖于其他属性被独立理解；④定义的属性在问题空间是否与这种实例关系密切；⑤定义的属性在分类结构中的位置是否恰当，底层对象的共有属性是否在上层对象属性体现；⑥在问题空间中每个对象的属性是否定义完整；⑦定义的实例关联是否符合现实；⑧在问题空间中实例关联是否定义完整，特别需要注意一对多和多对多的实例关联。

5. 对定义的服务和消息关联的测试

定义的服务，就是定义的每一种对象和结构在问题空间所要求的行为。由于问题空间中实例间需要通信，在 OOA 中需要为它们定义消息关联。对定义的服务和消息关联的测试可从以下几方面进行：①对象和结构在问题空间的不同状态是否定义了相应的服务；②对象或结构所需要的服务是否都定义了相应的消息关联；③定义的消息关联所指引的服务提供是否正确；④沿着消息关联执行的线程是否合理，是否符合现实过程；⑤定义的服务是否重复，是否定义了能够得到服务。

13.2.2 面向对象设计的测试

通常的结构化设计方法，用的"是面向作业的设计方法，它把系统分解以后，提出一组作

业，这些作业是以过程实现系统的基础构造，把问题域的分析转化为求解域的设计，分析的结果是设计阶段的输入”。

而面向对象设计(OOD)采用“造型的观点”，以 OOA 为基础归纳出类，并建立类结构或进一步构造成类库，实现分析结果对问题空间的抽象。OOD 归纳的类，可以是对象简单的延续，可以是不同对象的相同或相似的服务。由此可见，OOD 不是在 OOA 上的另一思维方式的大动干戈，而是 OOA 的进一步细化和抽象。所以，OOD 与 OOA 的界限通常是难以严格区分的。OOD 确定类和类结构不仅满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，能方便实现功能的重用和扩增，以不断适应用户的要求。因此，对 OOD 的测试，建议针对功能的实现和重用，以及对 OOA 结果的拓展，从三方面考虑：①对认定的类的测试；②对构造的类层次结构的测试；③对类库的支持的测试。

1. 对认定的类的测试

OOD 认定的类可以是 OOA 中认定的对象，也可以是对象所需要的服务的抽象、对象所具有的属性的抽象。认定的类原则上应该尽量基础性，这样才便于维护和重用。测试认定的类：①是否涵盖了 OOA 中所有认定的对象；②是否能体现 OOA 中定义的属性；③是否能实现 OOA 中定义的服务；④是否对应着一个含义明确的数据抽象；⑤是否尽可能少地依赖其他类；⑥类中的方法(C++：类的成员函数)是否单用途。

2. 对构造的类层次结构的测试

为能充分发挥面向对象的继承共享特性，OOD 的类层次结构通常基于 OOA 中产生的分类结构的原则来组织，着重体现父类和子类间的一般性和特殊性。在当前的问题空间，对类层次结构的主要要求是能在解空间构造实现全部功能的结构框架。为此，测试以下几方面：①类层次结构是否涵盖了所有定义的类；②是否能体现 OOA 中所定义的实例关联；③是否能实现 OOA 中所定义的消息关联；④子类是否具有父类中没有的新特性；⑤子类间的共同特性是否完全在父类中得以体现。

3. 对类库的支持的测试

对类库的支持虽然也属于类层次结构的组织问题，但其强调的重点是再次软件开发的重用。由于它并不直接影响当前软件的开发和功能实现，因此，将其单独提出来测试，也可作为对高质量类层次结构的评估。拟定的测试点有：①一组子类中关于某种含义相同或基本相同的操作，是否有相同的接口(包括名字和参数表)；②类中方法(C++：类的成员函数)功能是否较单纯，相应的代码行是否较少(建议不超过 30 行)；③类的层次结构是否深度大，宽度小。

13.2.3　面向对象编程的测试

典型的面向对象程序具有继承、封装和多态的新特性，这使得传统的测试策略必须有所改变。封装是对数据的隐藏，外界只能通过被提供的操作来访问或修改数据，这样降低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试。继承是面向对象程序的重要特点，继承使得代码的重用率提高，同时也使错误传播的概率提高。继承使得传统测试遇见了这样一个难题：对继承的代码究竟应该怎样测试？(参见面向对象单元测试)多态使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内“同一”函数的行为复杂化，测试时不得不考虑不同类型具体执行的代码和产生的行为。

面向对象程序是把功能的实现分布在类中。能正确实现功能的类，通过消息传递来协同实现设计要求的功能。正是这种面向对象程序风格，将出现的错误精确地确定在某一具体的类。因此，在面向对象编程(OOP)阶段，忽略类功能实现的细则，将测试的目光集中在类功能

的实现和相应的面向对象程序风格，主要体现在两个方面(假设编程使用 C++ 语言)：①数据成员是否满足数据封装的要求；②类是否实现了要求的功能。

1. 数据成员是否满足数据封装的要求

数据封装是数据和数据有关的操作的集合。检查数据成员是否满足数据封装的要求，基本原则是数据成员是否被外界(数据成员所属的类或子类以外的调用)直接调用。更直观地说，当改变数据成员的结构时，是否影响了类的对外接口，是否会导致相应外界必须改动。值得注意的是，有时强制的类型转换会破坏数据的封装特性。例如：

```
Class Hiden
{
      Private:
          int a = 1;
          char * p = &quot;hidden&quot;;
}
Class Visible
{
     Public:
         int b = 2;
         char * s = &quot;visible&quot;;
}
…
…
Hiden pp;
Visible * qq = (Visible * )&amp;#38;pp;
```

在上面的程序段中，pp 的数据成员可以通过 qq 被随意访问。

2. 类是否实现了要求的功能

类所实现的功能，都是通过类的成员函数执行。在测试类的功能实现时，应该首先保证类成员函数的正确性。单独地看待类的成员函数，与面向过程程序中的函数或过程没有本质的区别，几乎所有传统的单元测试中所使用的方法，都可在面向对象的单元测试中使用(具体的测试方法会在面向对象的单元测试部分介绍)。类函数成员的正确行为只是类能够实现要求功能的基础，类成员函数间的作用和类之间的服务调用是单元测试无法确定的。因此，需要进行面向对象的集成测试(具体的测试方法会在面向对象的集成测试部分介绍)。需要着重声明，测试类的功能，不能仅满足于代码能无错运行或被测试类能提供的功能无错，应该以所做的 OOD 结果为依据，检测类提供的功能是否满足设计的要求，是否有缺陷。必要时(如通过 OOD 结果仍不清楚明确的地方)还应该参照 OOA 的结果，以之为最终标准。

13.2.4 面向对象的单元测试

传统的单元测试是针对程序的函数、过程或完成某一定功能的程序块。沿用单元测试的概念，面向对象的单元测试实际测试类成员函数。一些传统的测试方法在面向对象的单元测试中都可以使用，如等价类划分法、因果图法、边界值分析法、逻辑覆盖法、路径分析法、程序插桩法等。

面向对象软件的单元概念发生了变化。封装驱动了类和对象的定义，这意味着每个类和类的实例(对象)包装了属性(数据)和操纵这些数据的操作(也称为方法或服务)，而不是个体的模块。最小的可测试单位是封装的类或对象，类包含一组不同的操作，并且某特殊操作可能作为一组不同类的一部分存在。因此，单元测试的意义发生了较大变化。

我们不再孤立地测试单个操作(传统的单元测试观点)，而是将操作作为类的一部分。对

OO 软件的类测试等价于传统软件的单元测试。和传统软件的单元测试不一样，它往往关注模块的算法细节和模块接口间流动的数据，OO 软件的类测试是由封装在类中的操作和类的状态行为所驱动的。

用于单元级测试的测试分析（提出相应的测试要求）和测试用例（选择适当的输入，达到测试要求）的规模和难度等远小于后面将介绍的对整个系统的测试分析和测试用例，而且强调对语句应该有 100％的执行代码覆盖率。在设计测试用例选择输入数据时，可以基于这两个假设：①如果函数（程序）对某一类输入中的一个数据正确执行，则对同类中的其他输入也能正确执行；②如果函数（程序）对某一复杂度的输入正确执行，则对更高复杂度的输入也能正确执行。例如需要选择字符串作为输入时，基于本假设，就无须计较字符串的长度。除非字符串的长度是要求固定的，如 IP 地址字符串。

在面向对象程序中，类成员函数通常都很小，功能单一，函数间调用频繁，容易出现一些不宜发现的错误。例如：

```
if ( -1 == write ( fid,buffer,amount )) error_out();
```

该语句没有全面检查 write()的返回值，仅只考虑数据被完全写入和没有写入两种情况。如果测试也忽略了数据部分写入的情况，就会给程序遗留隐患：①按程序的设计，使用函数 strrchr()查找最后的匹配字符，但程序中误写成了函数 strchr()，使程序功能实现时查找的是第一个匹配字符；②程序中将 if(strncmp(str1，str2，strlen(str1)))误写成了 if(strncmp(str1，str2，strlen(str2)))，如果测试用例中使用的数据 str1 和 str2 长度一样，就无法检测出。

因此，在做测试分析和设计测试用例时，应注意面向对象程序的这个特点，仔细地进行测试分析和设计测试用例，尤其是针对以函数返回值作为条件判断选择字符串操作的情况。

面向对象编程的特性使得对成员函数的测试，又不完全等同于传统的函数或过程测试。尤其是继承特性和多态特性，使子类继承或重载的父类成员函数出现了传统测试中未遇见的问题。Brian Marick 给出了两方面的考虑。

1. 继承的成员函数是否都不需要测试

对父类中已经测试过的成员函数，下面两种情况需要在子类中重新测试：继承的成员函数在子类中做了改动；成员函数调用了改动过的成员函数。例如：假设父类 Bass 有两个成员函数 Inherited()和 Redefined()，子类 Derived 只对 Redefined()做了改动。Derived::Redefined()显然需要重新测试。对于 Derived::Inherited()，如果它有调用 Redefined()的语句(如：x=x/Redefined())，就需要重新测试，反之，无此必要。

2. 对父类的测试是否能照搬到子类

延用上面的假设，Base::Redefined()和 Derived::Redefined()已经是不同的成员函数，它们有不同的服务说明和执行。对此，照理应该对 Derived::Redefined()重新测试分析，设计测试用例。但由于面向对象的继承使得两个函数又相似，故只需在 Base::Redefined()的测试要求和测试用例上添加对 Derived::Redfined()新的测试要求和增补相应的测试用例即可。

例如：

Base::Redefined()含有如下语句

```
If (value & lt;0) message (&quot ; less & quot;);
else if (value == 0) message (&quot ; equal & quot;);
else message (&quot ; more & quot;);
```

Derived：Redfined()中定义为

```
If (value & lt ; 0) message (&quot ;less & quot;);
```

```
else if (value == 0) message (&quot ; It is equal &quot;);
else
{
   Message (&quot ; more & quot;);
   if (value == 88) message (&quot ; luck & quot;);
}
```

在原有的测试上，只需对 Derived::Redfined()的测试做如下改动：在 value==0 的测试期望结果的基础上，增加 value==88 的测试。

多态有几种不同的形式，如参数多态、包含多态、重载多态。包含多态和重载多态在面向对象语言中通常体现在子类与父类的继承关系，对这两种多态的测试参见上述对父类成员函数继承和重载的论述。包含多态虽然使成员函数的参数可有多种类型，但通常只是增加了测试的繁杂。对具有包含多态的成员函数测试时，只需要在原有的测试分析和基础上扩大测试用例中输入数据的类型的考虑。

13.2.5 面向对象的集成测试

传统的集成测试，是自底向上通过集成完成的功能模块进行测试，一般可以在部分程序编译完成的情况下进行。而对于面向对象程序，相互调用的功能散布在程序的不同类中，类通过消息相互作用来申请和提供服务。类的行为与它的状态密切相关，状态不仅仅是体现在类数据成员的值，也许还包括其他类中的状态信息。由此可见，类相互依赖极其紧密，根本无法在编译不完全的程序上对类进行测试。所以，面向对象的集成测试通常需要在整个程序编译完成后进行。此外，面向对象程序具有动态特性，程序的控制流往往无法确定，因此也只能对整个编译后的程序做基于"黑盒"的集成测试。

面向对象的集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。基于单元测试能够确保成员函数正确性的前提下，集成测试只关注于系统的结构和内部的相互作用。面向对象的集成测试可以分成两步进行：先进行静态测试，再进行动态测试。

静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求。现在流行的一些测试软件都提供了程序理解的功能，即通过原程序得到类关系图和函数功能调用关系图。将程序理解得到的结果与 OOD 的结果相比较，检测程序结构和实现上是否有缺陷。换句话说，通过这种方法检测 OOP 是否达到了设计要求。

动态测试设计测试用例时，通常需要以上述的功能调用结构图、类关系图或者实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，减少测试工作量，使得进行的测试能够达到一定覆盖标准。测试所要达到的覆盖标准可以是：达到类所有的服务要求或服务提供的一定覆盖率；依据类间传递的消息，达到对所有执行线程的一定覆盖率；达到类的所有状态的一定覆盖率等。同时也可以考虑使用现有的一些测试工具来得到程序代码执行的覆盖率。

具体设计测试用例时，可参考以下几个步骤：①选定检测的类，参考 OOD 分析结果，仔细检测出类的状态和相应的行为、类或成员函数间传递的消息、输入或输出的界定等；②确定覆盖标准；③利用结构关系图确定待测类的所有关联；④根据程序中类的对象构造测试用例，确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

值得注意的是，设计测试用例时，不但要设计确认类功能满足的输入，还应该有意识地设计一些被禁止的例子，确认类是否有不合法的行为产生，如发送与类状态不相适应的消息或与要求不相适应的服务等。根据具体情况，动态地集成测试，有时也可以通过系统测试来完成。

13.2.6 面向对象的系统测试

通过单元测试和集成测试,仅能保证软件开发的功能得以实现,而不能确认在实际运行时,它是否满足用户的需要,是否大量存在实际使用条件下会被诱发产生错误的隐患。为此,对完成开发的软件必须进行规范的系统测试。换个角度说,开发完成的软件仅仅是实际投入使用系统的一个组成部分,需要测试它与系统其他部分配套运行的表现,以保证在系统各部分协调工作的环境下也能正常工作。

系统测试应该尽量搭建与用户实际使用环境相同的测试平台,应该保证被测系统的完整性,对临时没有的系统设备部件,也应有相应的模拟手段。系统测试时,应该参考OOA分析的结果,对应描述的对象、属性和各种服务,检测软件是否能够完全"再现"问题空间。系统测试不仅是检测软件的整体行为表现,从另一个侧面看,也是对软件开发设计的再确认。

系统测试是对所有类和主程序构成的整个系统进行整体测试,以验证软件系统的正确性和性能指标等满足需求规格说明书和任务书所指定的要求。它体现的具体测试内容如下。

(1) 功能测试:测试是否满足开发要求,是否能够提供设计所描述的功能,用户的需求是否都得到满足。功能测试是系统测试最常用和必需的测试,通常还会以正式的软件说明书为测试标准。

(2) 强度测试:测试系统的能力最高实际限度,即软件在一些超负荷情况下的功能实现情况。如要求软件某一行为的大量重复、输入大量的数据或大数值数据、对数据库大量复杂的查询等。

(3) 性能测试:测试软件的运行性能。这种测试常常与强度测试结合进行,需要事先对被测软件提出性能指标,如传输连接的最长时限、传输的错误率、计算的精度、记录的精度、响应的时限和恢复时限等。

(4) 安全测试:验证安装在系统内的保护机制确实能够对系统进行保护,使之不受各种非常的干扰。安全测试时需要设计一些测试用例试图突破系统的安全保密措施,检验系统是否存在安全保密漏洞。

(5) 恢复测试:采用人工的干扰使软件出错,中断使用,检测系统的恢复能力,特别是通信系统。恢复测试时,应该参考性能测试的相关测试指标。

(6) 可用性测试:测试用户是否能够满意使用。具体体现为操作是否方便,用户界面是否友好等。

(7) 安装/卸载测试(Install/Uninstall Test)。

系统测试需要对被测的软件结合需求分析做仔细的测试分析,建立测试用例。测试用例可从对象—行为模型和作为面向对象分析的事件流图中导出。

13.2.7 面向对象软件的回归测试

面向对象软件的回归测试不再作为测试的一个独立的阶段,而是以增量方式进行的。对象系统中的交互,既发生在类内的方法间又发生在多个类之间。如类A与类B的交互可通过:①B的实例变量作为参数传给类A的某方法。这时,类B的改变必然导致对类A的方法的回归测试。如果类A的方法又与其他方法发生交互,这些方法及其类就必须都进行回归测试。②类A的实例是类B表示的一部分。这时,B的所有对类A中变量进行引用的方法必须进行回归测试。

在这两种情况下,测试依赖集可用来确定第一种情况中所有与方法A的交互集;对象的

数据依赖集则可用来确定上述两种情况中所有必须重新测试的类和方法。

对面向对象软件的修改主要有：①不修改对象属性，只对方法进行修改。该对象以及系统中所有与该对象发生交互的对象都要进行完全新的测试。通常，这种测试代价是很高的。②修改对象属性。

13.2.8 基于UML的面向对象软件测试

统一建模语言(UML)在1997年被国际标准化组织(OMG)接纳为正式官方标准后已成为新一代面向对象软件设计的事实标准，其与IBM Rational统一过程(或其他软件工程开发过程)的配合使用更是被业界许多企业所采用。

UML提供了一套描述软件系统模型的概念和图形表示方法，以及语言的扩展机制和对象约束语言，软件开发人员可以使用UML对复杂的面向对象软件系统建立可视化的模型，并通过增量式的不断细化直接控制从设计、编码、测试到文档编制的整个软件开发过程。

UML为使用者提供了丰富的图形表示方式，使得使用者可以从不同的抽象角度对软件系统的特征进行描述。UML是一种图形化的设计语言，它使用不同类型的图从不同的角度和抽象层次上描述系统模型。在使用UML进行软件设计建模时，设计者必须根据所需要描述的对象和系统动作选择适当的UML图，以达到设计效果。同样在软件测试过程中，在不同的测试阶段，根据不同的测试目的，必须选择不同的UML图。以下将分析几种常用的UML图形的特点及其对软件测试的影响。

1. 类图

类图展现了一组对象、接口、协作和它们之间的关系。类图是面向对象建模中最为常见的图形。如图13-3所示，类图给出了系统的静态设计视图(包含主动类的类图)以及静态进程图。好的类图不仅能够描述系统中的类和这些类之间的关系，而且可以针对方法层的类和对象来引入附加的属性和关系。由于类图是对面向对象系统中类的最为详尽和全面的描述方式，因而能够帮助测试者全面掌握系统中的类结构，这对于类的测试有很大的帮助。

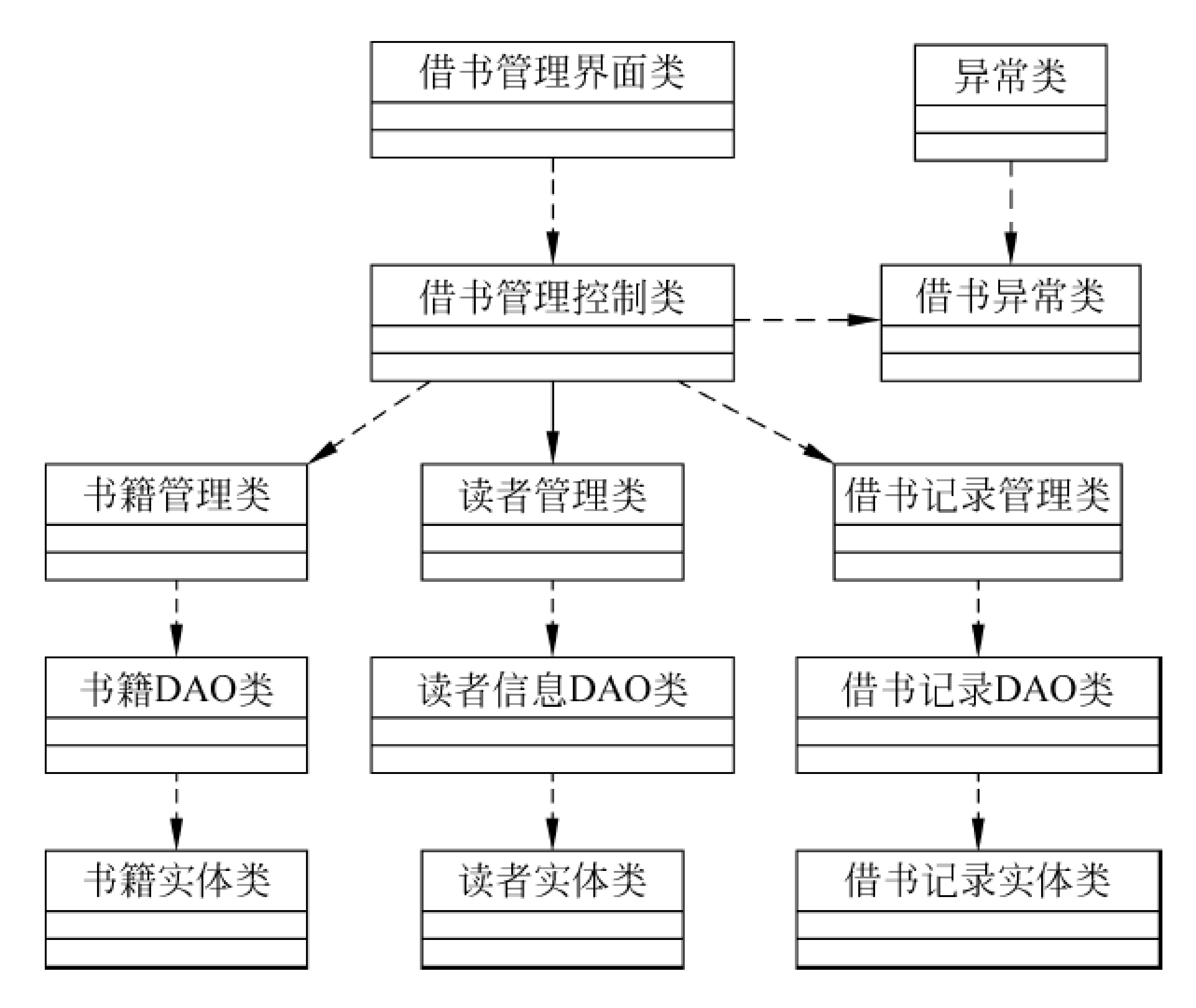


图13-3 图书馆管理系统的类图

2. 用例图

如图13-4所示，用例图展现了一组用例、参与者以及它们之间的关系。它给出了系统的

静态用例视图。用例(Use Case)图往往是在一个较高的抽象层次上描述系统的行为和各个组件之间的关系,所以能够帮助测试者全面把握系统的运行情况,是集成测试和系统测试的重要参考工具。

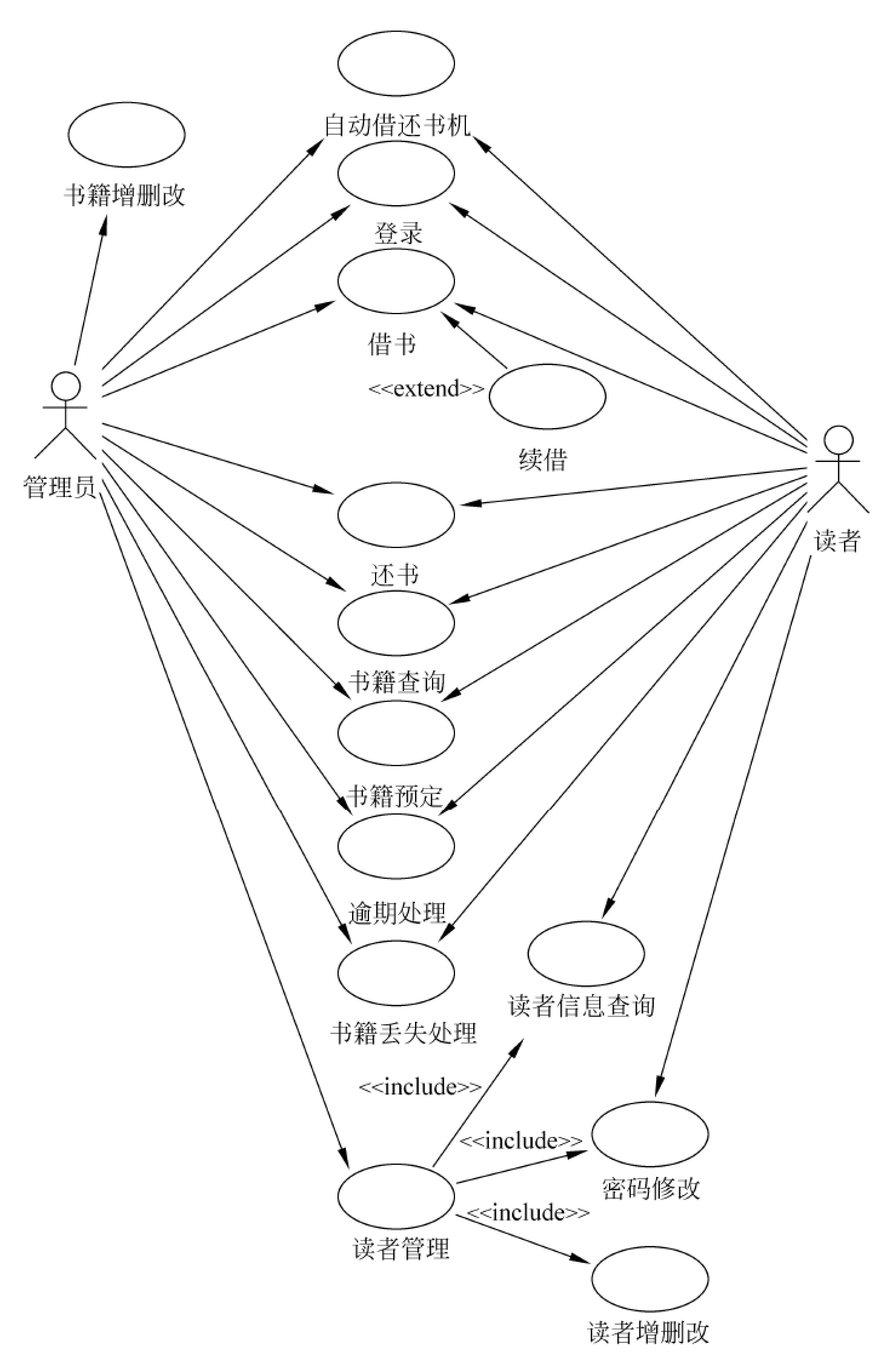


图 13-4 图书馆管理系统的用例图

3. 状态图

如图 13-5 所示,状态图展现了一个状态机,它由状态、转换、事件和活动组成,是专注于系统的动态视图。由于状态图强调对象行为的事件顺序,因而它对于接口、类和协作的测试都有很重要的作用。由于建立在状态机的基础上,所以很多已有的对于状态机测试的有效方法都能很容易地被移植到对应状态图的测试中去。通常采用 W-方法构造所需的测试用例,或者将状态机转换为数据流图,依据数据流分析方法构造测试用例。

4. 序列图

如图 13-6 所示,序列图是一种强调消息的时间顺序的交互图,主要用来设计和描述算法。

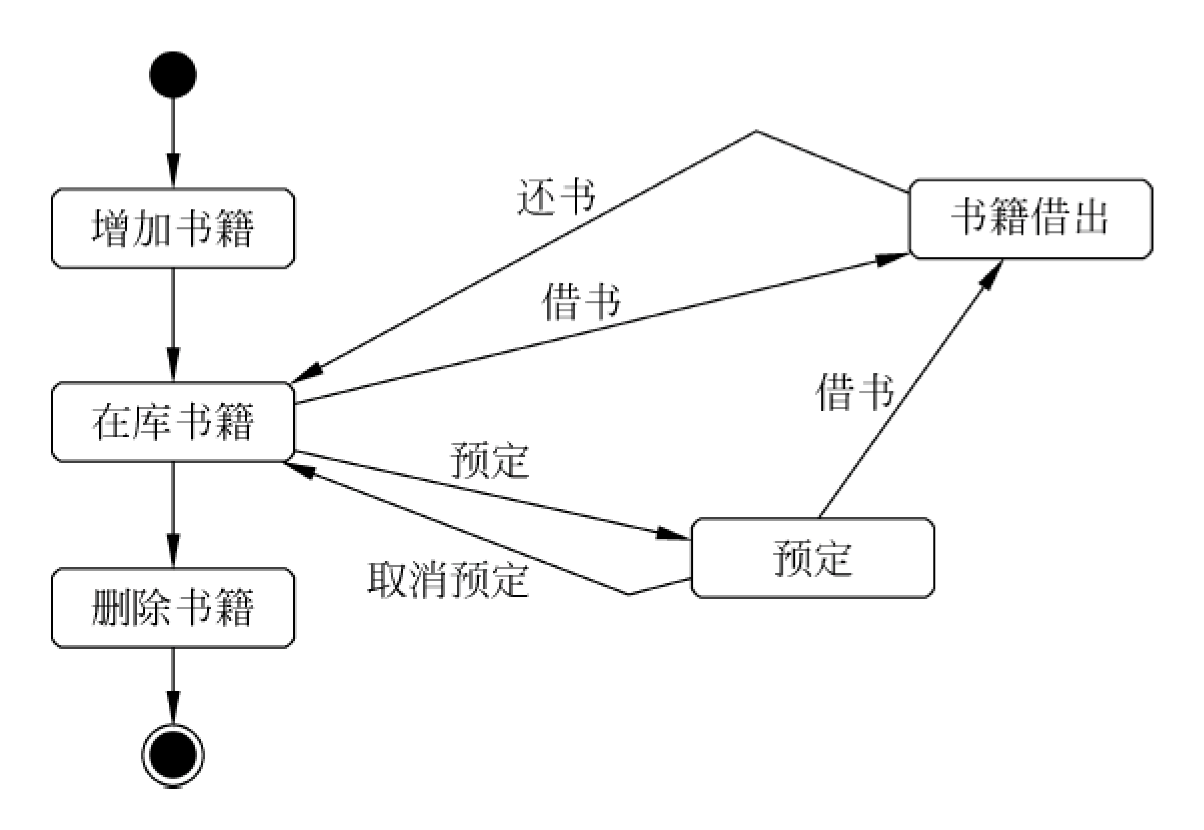


图 13-5 图书馆的书籍状态图

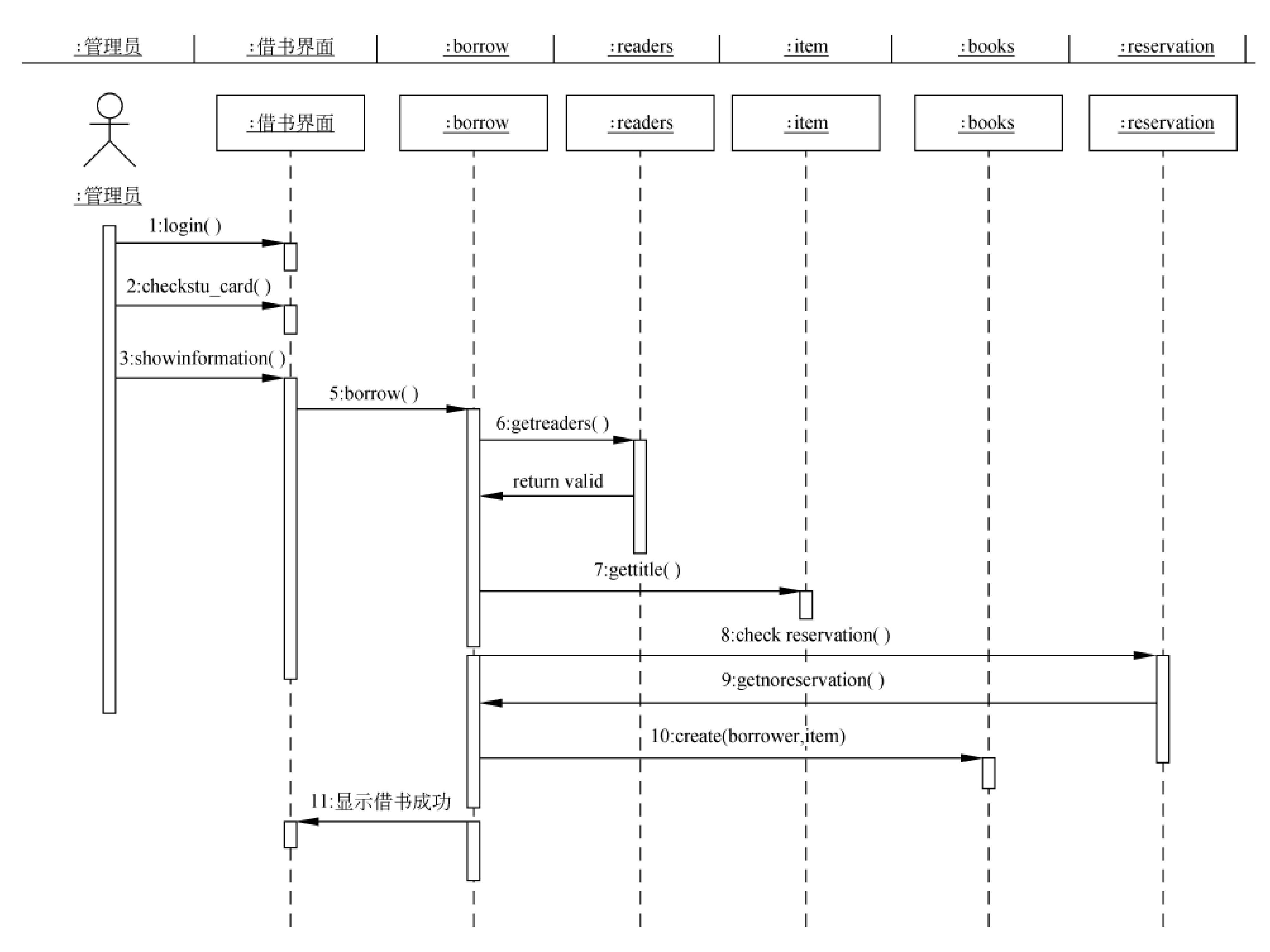


图 13-6 图书馆管理系统借书序列图

从测试的角度来看，序列图中可能存在一些错误，其中包括约定的冲突、不能创建正确的对象、图中没有关系的发送者和接收者之间的消息传递等。

在 9 种 UML 图中，把用于对系统的静态方面进行可视化、详述、构造和文档化的 UML 图称为结构图(参见表 13-2)。

表 13-2 结构图的名称和描述对象

名 称	描述对象	名 称	描述对象
类图	类、接口、协作	构建图	构建
对象图	对象	实施图	节点

同样，可以把用于对系统的动态方面进行可视化、详述、构造和文档化的 UML 图称为行为图(参见表 13-3)。

表 13-3　行为图的名称和描述对象

名　称	描 述 对 象	名　称	描 述 对 象
用例图	组织系统的行为	状态图	有事件驱动和系统的变化状况
序列图	消息的时间顺序	活动图	从活动到活动的控制流
协作图	收发消息的对象的结构组织		

根据 UML 图的不同抽象层次，在软件开发和测试的不同阶段使用的 UML 图也各不相同(参见表 13-4)。

表 13-4　UML 图与软件设计和测试的对应关系

软件设计	UML 图	软件测试
需求分析	用例图 实施图	确认测试
功能划分	活动图 构建图	系统测试
系统设计	状态图 对象图 协作图	集成测试
编码实现	状态图 对象图 协作图 序列图 类图	单元测试

13.3　面向对象软件测试用例的设计

传统软件测试用例设计是从软件的各个模块的算法细节得出的，而面向对象的软件测试用例则着眼于适当的操作序列，以实现对类的说明。

“黑盒”测试不仅适用于传统软件，也适用于面向对象的软件测试。同样，“白盒”测试也适用于面向对象的软件类的操作定义，但面向对象的软件中许多类的操作结构简明，所以有人认为在类层上测试可能要比传统软件中的“白盒”测试方便。

13.3.1　基于故障的测试

在面向对象的软件中，基于故障的测试具有较高的发现可能故障的能力。由于系统必须满足用户的需求，因此，基于故障的测试要从分析模型开始，考察可能发生的故障。为了确定这些故障是否存在，可设计用例去执行设计或代码。

基于故障测试的关键取决于测试设计者如何理解“可能的错误”。而在实际中，要求设计者做到这点是不可能的。基于故障测试也可以用于集成测试，集成测试可以发现消息联系中“可能的故障”。“可能的故障”一般为意料之外的结果、错误地使用了操作、消息不正确引用等。为了确定由操作(功能)引起的可能故障，必须检查操作的行为。这种方法除用于操作测试外，还可用于属性测试，用以确定其对于不同类型的对象行为是否赋予了正确的属性值(因为一个对象的“属性”是由其赋予属性的值定义的)。

13.3.2　基于脚本的测试

基于脚本的测试主要关注用户需要做什么，而不是产品能做什么，即从用户任务(使用用例)中找出用户要做什么并去执行。这种基于脚本的测试有助于在一个单元测试情况下检查多重系统。所以基于脚本测试比基于故障测试不仅更实际(接近用户)，而且更复杂一点。

13.3.3 面向对象类的随机测试

如果一个类有多个操作(功能),这些操作(功能)序列有多种排列,而这种不变化的操作序列可随机产生,用这种可随机排列的序列来检查不同类实例的生存史,就叫随机测试。

例如一个银行信用卡的应用,其中有一个类:计算(account)。该 account 的操作有:open、setup、deposit、withdraw、balance、sum-marize、creditlimit 和 close。

这些操作中的每一项都可用于计算,但 open 和 close 必须在其他计算的任何一个操作前后执行,即使 open 和 close 有这种限制,这些操作仍有多种排列。所以一个不同变化的操作序列会因应用不同而随机产生,如一个 account 实例的最小行为生存史可包括以下操作:

open + setup + deposit + [deposit|withdraw|balance|summarize|creditlimit] + withdraw + close

由此可见,尽管这个操作序列是最小测试序列,但在这个序列内仍可以发生许多其他的行为。

1. 类层次的分割测试

类层次的分割测试可以减少用完全相同的方式检查类测试用例的数目。这很像传统软件测试中的等价类划分测试。分割测试又可分三种:①基于状态的分割,按类操作是否改变类的状态来分割(归类);②基于属性的分割,按类操作所用到的属性来分割(归类);③基于类型的分割,按完成的功能分割(归类)。

2. 由行为模型(状态、活动、顺序和合作图)导出的测试状态转换图(STD)

操作序列可用来帮助导出类的动态行为的测试序列,以及这些类与合作类的动态行为测试序列。

为了说明问题,仍用前面讨论过的 account 类。开始由 empty acct 状态转换为 setup acct 状态。类实例的大多数行为发生在 working acct 状态。而最后,取款和关闭分别使 account 类转换到 non-working acct 和 dead acct 状态。这样,设计的测试用例应当是完成所有的状态转换。换句话说,操作序列应当能导致 account 类所有允许的状态进行转换。

测试用例:open+setup acct+deposit(initial)+withdraw(final)+close。还可导出更多的测试用例,以保证该类所有行为被充分检查。

习题

1. 什么是面向对象技术?它和结构化程序设计方法相比有哪些特点?
2. 传统的软件测试策略是什么?为什么不适用于面向对象软件?
3. 为什么面向对象编程语言的有关语言特征会对软件测试产生影响?
4. 面向对象软件测试包括哪些内容?如何进行面向对象软件的测试?

第14章 客户端应用软件测试

客户端(Client)或称为用户端,是指与服务器相对应,为客户提供本地服务的程序。除了一些只在本地运行的应用程序之外,一般安装在普通的客户机上,需要与服务端互相配合运行。因特网发展以后,较常用的用户端包括如万维网使用的网页浏览器、收寄电子邮件时的电子邮件客户端,以及即时通信的客户端软件等。对于这一类应用程序,需要网络中有相应的服务器和服务程序来提供相应的服务,如数据库服务、电子邮件服务等,这样在客户机和服务器端,需要建立特定的通信连接来保证应用程序的正常运行。

在计算机的世界里,凡是提供服务的一方称为服务端(Server),而接受服务的另一方称为客户端(Client)。我们最常接触的例子是局域网络里的打印服务器所提供的打印服务:提供打印服务的计算机,我们可以说它是打印服务器;而使用打印服务器提供打印服务的另一方,我们则称做客户端。但是谁是客户端谁是服务端也不是绝对的,例如倘若原提供服务的服务端要使用其他机器所提供的服务,则所扮演的角色即转变为客户端,而这种关系在因特网上,就变成使用者和网站的关系了。使用者透过调制解调器等设备上网,在浏览器中输入网址,透过 HTTP 通信协议向网站提出浏览网页的要求(Request)。网站收到使用者的要求后,将使用者要浏览的网页数据传输给使用者,这个动作称为响应(Response)。网站提供网页数据的服务,使用者接受网站所提供的数据服务,所以使用者在这里就是客户端,响应使用者要求的网站即称为服务端。

不过客户端及服务端的关系不见得一定建立在两台分开的机器上,同一台机器中也有这种主从关系的存在,提供服务的服务端及接受服务的客户端也有可能都在同一台机器上,例如我们在提供网页的服务器上执行浏览器浏览本机所提供的网页,这样在同一台机器上就同时扮演服务端及客户端。

随着计算机网络的发展,连接形式逐渐发生变化,至今为止,主流的两种网络连接模式为 C/S 和 B/S 模式,即客户端/服务器端和浏览器/服务器端。事实上,浏览器也是客户端,只不过它是一个特殊的客户端。

14.1 C/S、B/S 应用模式概述

14.1.1 C/S、B/S 技术特点

要想对“C/S”和“B/S”技术发展变化有所了解,首先必须搞清楚三个问题。

1. C/S 结构

C/S(Client/Server)结构(如图 14-1 所示),即大家熟知的客户机和服务器结构。它是软件系统体系结构,通过它可以充分利用两端硬件环境的优势,将任务合理分配到客户端和服务

14.1.2 C/S 和 B/S 的比较

C/S 和 B/S 是当今世界开发模式技术架构的两大主流技术。C/S 是美国 Borland 公司最早研发，B/S 是美国微软公司研发。目前，这两项技术已被世界各国所掌握，国内公司以 C/S 和 B/S 技术开发出的产品也很多。这两种技术都有自己一定的市场份额和客户群。

1. C/S 架构软件的优势与劣势

C/S 架构软件的优势与劣势如下。

(1) 应用服务器运行数据负荷较轻。最简单的 C/S 体系结构的数据库应用由两部分组成，即客户应用程序和数据库服务器程序。两者可分别称为前台程序与后台程序。运行数据库服务器程序的机器，也称为应用服务器。一旦服务器程序被启动，就随时等待响应客户程序发来的请求；客户应用程序运行在用户自己的电脑上，对应于数据库服务器，可称为客户电脑，当需要对数据库中的数据进行任何操作时，客户程序就自动地寻找服务器程序，并向其发出请求，服务器程序根据预定的规则做出应答，送回结果，应用服务器运行数据负荷较轻。

(2) 数据储存管理功能较为透明。在典型的 C/S 数据库应用中，数据的储存管理功能，是由服务器程序独立进行的，并且通常把那些不同的(不管是已知还是未知的)前台应用所不能违反的规则，在服务器程序中集中实现，例如访问者的权限，编号不准重复、必须有客户才能建立定单这样的规则。所有这些，对于工作在前台程序上的最终用户，是“透明”的，他们无须过问(通常也无法干涉)这背后的过程，就可以完成自己的一切工作。在客户服务器架构的应用中，前台程序可以变得非常“瘦小”，麻烦的事情，都交给了服务器和网络。在 C/S 体系下，数据库真正变成了公共、专业化的仓库，受到独立的专门管理。

(3) C/S 架构的劣势是高昂的维护成本且投资大。首先，采用 C/S 架构，要选择适当的数据库平台来实现数据库数据的真正“统一”，使分布于两地的数据同步完全交由数据库系统去管理，但逻辑上两地的操作者要直接访问同一个数据库才能有效实现，有这样一些问题，如果需要建立“实时”的数据同步，就必须在两地间建立实时的通信连接，保持两地的数据库服务器在线运行，网络管理工作人员既要对服务器维护和管理，又要对客户端维护和管理，这需要高昂的投资和复杂的技术支持，维护成本很高，维护任务量大。

其次，传统的 C/S 结构的软件需要针对不同的操作系统开发不同版本的软件，由于产品的更新换代十分快，代价高和低效率，已经不适应工作需要。在 Java 这样的跨平台语言出现之后，B/S 架构更是猛烈冲击 C/S，并对其形成威胁和挑战。

2. B/S 架构软件的优势与劣势

B/S 架构软件的优势与劣势如下。

(1) 维护和升级方式简单。目前，软件系统的改进和升级越来越频繁，B/S 架构的产品明显体现着更为方便的特性。对一个稍微大一点单位来说，系统管理人员如果需要在几百甚至上千部电脑之间来回奔跑，效率和工作量是可想而知的，但 B/S 架构的软件只需要管理服务器就行了，所有的客户端只是浏览器，根本不需要做任何的维护。无论用户的规模有多大，有多少分支机构都不会增加任何维护升级的工作量，所有的操作只需要针对服务器进行；如果是异地，只需要把服务器连接专网，即可实现远程维护、升级和共享。所以客户机越来越“瘦”，而服务器越来越“胖”，是将来信息化发展的主流方向。今后，软件升级和维护会越来越容易，而使用起来会越来越简单，这对用户人力、物力、时间、费用的节省是显而易见的，惊人的。因此，维护和升级革命的方式是“瘦”客户机，“胖”服务器。

(2) 成本降低，选择更多。大家都知道 Windows 在桌面电脑上几乎一统天下，浏览器成

为了标准配置，但在服务器操作系统上 Windows 并不是处于绝对的统治地位。现在的趋势是凡使用 B/S 架构的应用软件，只需安装在 Linux 服务器上即可，而且安全性高。所以服务器操作系统的选择是很多的，不管选用哪种操作系统都可以让大部分人使用 Windows 作为桌面操作系统电脑不受影响，这就使得最流行、免费的 Linux 操作系统快速发展起来，Linux 除了操作系统是免费的以外，连数据库也是免费的。

比如说很多人每天上“网易”，只要安装了浏览器就可以了，并不需要了解“网易”的服务器用的是什么操作系统，而事实上大部分网站确实没有使用 Windows 操作系统，但用户的电脑本身安装的大部分是 Windows 操作系统。

(3) 应用服务器运行数据负荷较重。由于 B/S 架构应用软件只安装在服务器端上，网络管理人员只需要管理服务器就行了，用户界面主要事务逻辑在服务器端，极少部分事务逻辑在前端实现。所有的客户端只有浏览器，网络管理人员只需要做硬件维护。但是，应用服务器运行数据负荷较重，一旦发生服务器“崩溃”等问题，后果不堪设想。因此，许多单位都备有数据库存储服务器，以防万一。

14.1.3 C/S 与 B/S 的区别

Client/Server 是建立在局域网的基础上的，Browser/Server 是建立在广域网的基础上的。

1. 硬件环境不同

C/S 一般建立在专用的网络上，小范围里的网络环境，局域网之间再通过专门服务器提供连接和数据交换服务。

B/S 建立在广域网之上，不必是专门的网络硬件环境，例如电话上网，租用设备，信息自己管理，有比 C/S 更强的适应范围，一般只要有操作系统和浏览器就行。

2. 对安全要求不同

C/S 一般面向相对固定的用户群，对信息安全的控制能力很强。一般高度机密的信息系统采用 C/S 结构适宜，可以通过 B/S 发布部分可公开信息。

B/S 建立在广域网之上，对安全的控制能力相对弱，面向不可知的用户群。

3. 对程序架构不同

C/S 程序可以更加注重流程，可以对权限多层次校验，对系统运行速度可以较少考虑。

B/S 对安全以及访问速度进行多重的考虑，建立在需要更加优化的基础之上。比 C/S 有更高的要求，B/S 结构的程序架构是发展的趋势，从 MS 的.NET 系列的 BizTalk2012、Exchange2013 等全面支持网络的构件搭建的系统，到 SUN 和 IBM 推的 JavaBean 构件技术等，使 B/S 更加成熟。

4. 软件重用不同

C/S 应用需要整体性考虑，构件的重用性不如在 B/S 要求下的构件的重用性好。

B/S 的多重结构，要求构件具有相对独立的功能，满足有关重用的要求。就如买来的餐桌可以再利用，而不是做在墙上的石头桌子。

5. 系统维护不同

C/S 应用整体性要求高，处理出现的问题以及系统升级难。

B/S 构件组成方面，可通过构件个别地更换，实现系统的无缝升级。系统维护开销减到最小，用户从网上自己下载安装就可以实现升级。

6. 处理问题不同

C/S 面向固定的用户，并且在相同区域、安全要求高，与操作系统相关。

B/S建立在广域网上，面向不同的用户群，分散地域，这是C/S无法做到的，与操作系统平台关系最小。

7. 用户接口不同

C/S多是建立在Windows平台上，表现方法有限，对程序员普遍要求较高。

B/S建立在浏览器上，有更加丰富和生动的表现方式与用户交流，并且开发难度低，降低开发成本。

8. 信息流不同

C/S程序一般是典型的中央集中式处理，交互性相对低。

B/S信息流向可变化，B—B、B—C、B—G等信息流向的变化，更像交易中心。

14.2　C/S系统测试

C/S系统具有用户界面图形化、设计面向对象性、数据存储分布性、控制并发性以及平台异构性等特性，这些新的特性为C/S系统的软件测试引入了一系列新的问题。

14.2.1　C/S系统测试对传统测试的影响

C/S系统测试对传统测试的影响体现在以下几个方面。

(1) 从程序的组织结构方面来讲，传统的测试技术不完全适用于C/S系统的测试。

例如，在功能测试中，传统程序的测试过程是，选定一组数据，交给被测程序处理，通过比较实际执行结果和预期执行结果，判断程序是否含有错误。因此，传统软件测试技术与过程式程序中数据和操作相分离的特点相适应，是从输入/处理/输出(Input/Process/Output)的角度检验一个函数或过程能否正确工作，本质上体现了冯·诺依曼计算机体系结构的特点。

(2) C/S系统和传统的软件系统不同。

传统的软件系统是对一序列数据操作的过程或函数的集合，而C/S系统是由相互协作而又彼此之间相对独立的软件子系统构成的，各个子系统之间通过通信协议通信，子系统内部通过消息进行通信。

(3) 从体系结构上来讲，C/S系统由多层软件体系结构构成。

C/S系统一般由客户端子系统、应用服务子系统和数据库服务子系统构成，简单的C/S系统至少包括客户端子系统和数据库服务子系统。

(4) 从数据存储来讲，C/S系统的数据存储一般通过大型的关系式数据库管理系统(Oracle，SQL Server，DB2等)或面向对象的数据库管理系统。

通常Client子系统通过Server子系统访问数据库，Server子系统通过数据库引擎访问数据库中的数据。

(5) 从实现技术上讲，C/S系统的实现一般采用图形用户界面、面向对象技术。

从开发方法上，一般采用快速应用开发(Rapid Application Development，RAD)和共同应用开发(Joint Application Development，JAD)。

(6) 从运行的平台来讲，系统既可运行在相同的平台上，也可运行在不同的平台上。

尽管传统的测试技术仍然可以应用于C/S系统局部测试之中，但远远不能满足C/S系统软件测试的要求。因此，传统的测试技术必须经过改造才能实用于C/S系统的测试，同时，还需要专门研究针对C/S系统特点的测试理论和技术。

下面从C/S系统的体系结构、图形用户界面特性、面向对象特性以及开发等方面分别讨

论 C/S 系统特性对测试的影响。

1. 多层软件体系结构

C/S 系统的结构随着软件的发展而不断地扩展，从一个局域网（LAN）到广域网（WAN）以及和 Internet 互连而构成的网际网，都属于 C/S 结构或扩展的 C/S 结构的软件系统，其中和 Internet 互连构成的网络系统，也称 Browser/Server（简称 B/S）结构的系统。典型的 C/S 系统结构一般由客户端、服务端，以及用于客户端和服务端进行通信的中间协议三部分组成，如图 14-3 所示。

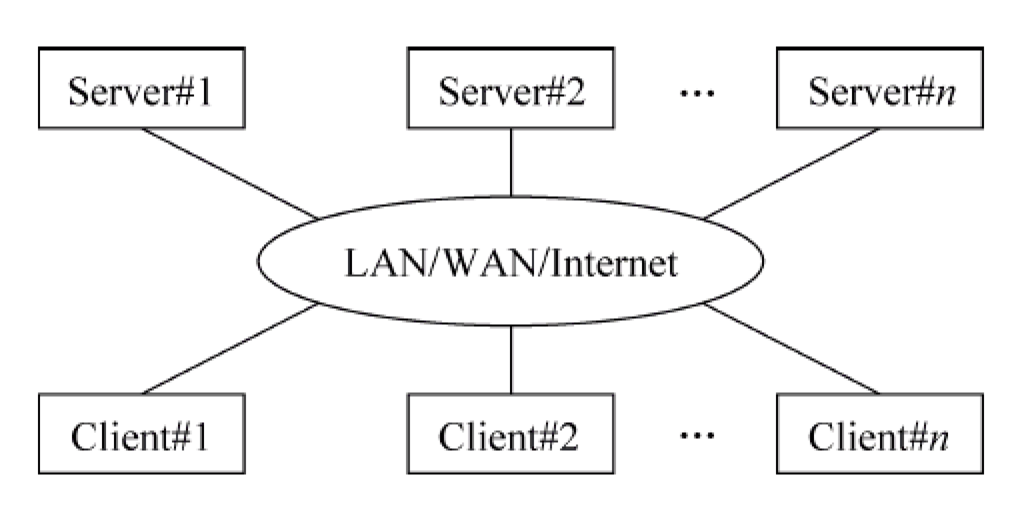


图 14-3　Client/Server 系统结构

C/S 系统采用多层软件体系结构的原因主要是：提高系统的性能；提高系统的可扩展性；提高系统的可维护性。然而，C/S 系统的多层体系结构给 C/S 软件系统的测试带来了很大的困难，必须对 C/S 系统的各个子系统进行测试，然后对整个系统进行集成测试。由于 C/S 系统由多层体系构成，各层之间存在各种复杂的关系，一个 C/S 系统的静态表示是一个复杂的网状结构，传统的系统集成测试策略已不适应 C/S 系统的集成测试。因此，需要研究适应 C/S 软件系统的特点的集成测试策略。另外，C/S 系统的客户端和服务端通过通信协议进行通信，需要研究通信协议一致性测试策略和方法。

2. 图形用户界面（GUI）

C/S 系统和传统的面向过程的系统的明显不同在于几乎所有的 C/S 系统都提供图形用户界面和用户进行交互。图形用户界面提供给用户显示在屏幕上的一系列的对象阵列（包括菜单、命令按钮、下拉列表、工具栏、滚动条等控件），对象之间互相依赖、互相影响。C/S 系统的图形用户界面给用户使用系统带来了极大的方便，用户可以非常容易而又直观地和系统进行交互，大大地减少了用户培训的时间和费用。然而，也给开发人员和测试人员增加了许多额外的困难。每个窗口上的每个对象都必须进行测试，更为复杂的是，窗口上的对象相互影响和相互控制，使得窗口上对象的测试工作成几何数量地增长。

C/S 系统的图形用户界面并非过程调用的，而是事件驱动的，图形用户界面系统是根据用户的产生事件（包括键盘动作、鼠标移动、按钮单击等）产生相应的响应。由于事件序列不可能预先知道，每个对象的事件数量多，并且各个对象的事件组合千变万化，给图形用户界面的测试带来了极大的困难。图形用户界面中的导航功能非常复杂，大多数系统采用菜单项选择、命令按钮或工具栏等实现图形用户界面的导航功能。图形用户界面的复杂导航功能要求对每条可以想象到的路径进行测试。

另外，图形用户界面中的多文档界面功能，每个文档界面是一个线程，极大地增加了路径测试和强度测试的困难。必须对系统进行大量的强度测试。综上所述，图形用户界面的特性给测试带来很大的困难，必须研究适合图形用户界面测试的技术和方法。

3. 面向对象

目前的 C/S 系统的实现大多都采用面向对象的程序设计（OOP）。面向对象的程序设计重要特征是信息隐蔽、封装和继承。在面向对象设计中将具有相同属性和操作的对象集合称为类。类的重要作用之一是信息隐蔽。它对类中所封装的信息直接进行控制，从而避免类中有关实现细节的信息被错误地使用。而这样的细节信息正是软件测试所不可忽略的。由于面向对象的软件系统在运行时刻由一组协调工作的对象组成，对象具有一定的状态，所以对面向对象的程序测试来说，对象的状态是必须考虑的因素，测试应涉及对象的初态、输入参数、输出

参数、对象的状态。对象的行为是被动的，它只有在接收有关消息后才被激活来进行所请求的工作，并将结果返回给发消息者。在工作过程中对象的状态可能被修改，产生新的状态。面向对象软件测试的基本工作就是创建对象（包括初始化），向对象发送一系列消息然后检查结果对象的状态，看其是否处于正确的状态。问题是对象的状态往往是隐蔽的，若类中未提供足够的存取函数来表明对象的实现方式和内部状态，则测试者必须增添这样的函数。因此，类的信息隐蔽机制给测试带来困难。

4. 平台异构

传统的软件系统一般运行在同构平台之上，而 C/S 系统一般运行在异构平台之上，数据库服务器可能是运行 UNIX 操作系统的 HP 服务器，应用服务器可能是 UNIX 服务器，也可能是基于 Intel 计算机的服务器。客户端既可以是各种基于 Pentium 的机器，也可以是各种 UNIX 的工作站。因此，C/S 系统在各种不同类型的机器上都必须进行充分的测试。

总之，C/S 系统测试因其计算与数据分布，导致并发和安全问题，使场景复杂；使用事件驱动和组件技术设计的 GUI 界面使得测试路径趋近无穷，测试场景复杂；使用对象编程技术使得对象之间的依赖和继承关系复杂，错误修改引起的连锁反应增大；使用对象和组件技术使得系统对第三方组件/类库依赖增强，在质量和技术上存在风险；系统本身复杂、多样，且使用了 RAD（快速应用软件开发）开发方式，导致文档内容复杂、文档不详细以及文档术语难以统一。另外，多系统、多语言使得错误的隐蔽性和数量增大，测试环境的搭建更加困难，测试人员的技术要求更加全面。如：由于数据库的使用，难于直接控制数据（数据独立要通过接口访问）。内置安全机制和应用层安全机制混在一起使得问题复杂；由于网络的使用，硬件之间和软件之间的通信通过网络和网络上面的协议来完成；由于多硬件、多软件、多数据库、多协议标准、多语言的混合使用，使得 C/S 系统失效、不匹配可能性增大；由于开发人员过多，开发团队过大，使得在软件系统开发过程中协调一致难度比较大。再就是，C/S 应用软件大量地使用和高度依赖于第三方系统，而这里面存在第三方产品的稳定性不能保证，多厂商带来的复杂性和管理问题（如厂商之间的版本影响，厂商之间的版本更新组合情况复杂，产品采购招标是一个总承包商，这样就造成厂商之间踢皮球等）。最后，针对特定的 C/S 应用软件进行测试的测试历史数据和针对性的测试方法匮乏，使得测试工作开展起来困难很多。因此，在这些因素的共同作用下，测试 C/S 系统比测试传统软件系统更加困难。

14.2.2　C/S 系统测试的目标

对 C/S 结构的系统进行测试，目的是：

（1）检查系统是否达到公布的功能说明。在系统进行开发前，系统的功能范围就已经在需求阶段定义好了，如果中途发生修改，则需要重新修改项目的计划和预算，系统的功能说明也需要随之逐步完善，尽可能地将用户的期望写入公布的功能说明。因此需要对系统进行测试，看其是否满足已公布的功能说明，尽早发现，付出的代价就越小。

（2）检查是否满足性能要求。相比于开发人员，用户永远更加关注系统的性能，因为除了系统的功能外，用户经常被困扰的问题是系统的性能；在系统进行交付时，有些性能问题可能会因为培训人员玩文字游戏而使得用户在当时无话可说，但是随着用户的使用，系统隐藏的性能问题也就逐步显现出来，因此使得用户的满意度下降、信任度下降，对公司的信誉造成很大的影响。例如：培训人员对客户说某个窗口在 1 秒内可用，但是实际上是只有 10％的窗口内容是在 1 秒钟内显示的，其他的内容还要再等一段时间才能显示。这样用户当时是很满意，但是等用户使用后，这个窗口的显示并没有预期的那么快，造成用户的容忍度下降，对产品的

块/产品之内,同一个系统之内是否一致;一些基本的界面元素(如按钮、表单、图片、文字等)的尺寸大小、边距、间距、布局是否和功能规格说明书相符;文字是否有错误拼写等。

(1) 检查本地化问题,如是否存在不支持本地化、部分本地化、本地化错误等问题;

(2) 检查信息表示问题,如是否存在信息模糊或者矛盾、信息显示不全等问题;

(3) 检查界面操作路径是否复杂、模糊等问题。

4) 负载/压力测试

负载/压力测试主要检验服务器端程序的负载效率,当大的客户访问量出现时,需要对服务器的执行效率进行模拟,以实现整个系统处理交易的能力,这也是为了保证系统的稳定性。在负载/压力测试过程中,一般需要一些辅助的测试工具进行模拟测试。

5) 安全性测试

安全性测试主要测试客户端程序、服务器端程序是否存在安全性漏洞,加密信息的保密性,防止黑客攻击能力等。

6) 兼容性测试

兼容性测试主要测试在各种操作系统(Windows、Mac OS、Solaris、Linux)上安装客户端程序是否能正常执行,在做兼容性测试时,需要对一些已知的因素进行规列(Windows、Mac OS 的限制等)。

7) 网络链接测试

网络链接测试主要测试不同的网络链接方式的速度、效率,同时需要考虑是否需要代理服务器,在代理服务器上怎样与客户端交互等。

8) 系统测试

系统测试主要测试那些存在于 C/S 系统之外,对 C/S 系统的运行产生影响的错误。例如:操作系统错误、中间件错误、DLL 错误、驱动程序错误、硬件错误、网络设备错误等。

系统测试的难点是很难隔离并确认错误发生的地点,而这导致供应商踢皮球,即使承认,解决问题也需要时间,并且会给系统带来新的不稳定。

因此,尽量在开始设计的时候考虑周全,并考察供应商资格和服务,绕过这个问题,请厂商修改系统,或更换厂商。

9) 通信错误

通信错误是指存在于 C/S 系统之外的,各个层之间通信问题产生的错误,包括硬件和同层。例如网卡损坏、电缆接触不良、通信软件或者驱动程序自身错误、用户权限不够、地址问题、路由器等通信设备损坏、私有协议错误等。

10) 数据错误

SQL 简单/强大,但是使用上有较大的风险,特别是涉及直接修改数据导致问题的产生。为此,要对开发人员进行 SQL 的培训,制定编码规范,互相检查代码,并请 SQL 专家把关。

建立 SQL 程序中的检查点,如是否检查了查询的返回错误值(包括 Select),仔细检查使用 Delete 和 Update 的地方,仔细检查存储过程和触发器、聚合函数的使用陷阱(不单独列出每一个记录),其他(如年龄的计算方法)。

建立对数据库的检查点,如 Schema 命名机制(变量作用域),安全性策略的设置和检查,多个数据库使用中日期表示的不同特点。

11) 编程错误,包括坏的编程习惯

编程错误与传统测试中遇到的问题一样,如变量初始化、变量名字类似/错误使用等,数据类型和移植问题(包括多系统一致性、计算能力迁移)。

12）其他方面的测试

其他方面的测试包括系统的不同分辨率下的界面显示、流量测试。

2. C/S 系统测试的常见测试点

在进行 C/S 系统测试时，我们经常重点关注如下 7 个方面的问题。

1）输入是否合法测试

不合法的输入常导致错误的发生，这是因为很多程序在代码中的错误处理分支、数据库中的约束、存储过程/触发器等方面存在问题。我们可以采用边界值测试方法；对日期，我们可以尝试各种时间的表示以及计算，如不同日期表示格式，闰年、星期几、时区、时制等计算。

2）路径测试或循环测试

类似于"白盒"测试技术中的路径测试或循环测试概念：对 C/S 系统进行完全路径测试是不现实的，我们可以使用基本测试路径方法。

3）事务测试

事务从设计角度来说是一个独立的工作单位，从数据库角度来说是一个全部执行/不执行的 SQL 集合，从用户角度来说是一个完全成功/取消的操作。在进行事务处理过程中经常会出现一些错误，如在一个表中修改记录，可能会引起多个表的更新；或删除一个主键可能会影响表关系的修改操作。这时我们在设计测试用例时可以分别考虑：

（1）输入合法的完整的记录，检查事务是否正确执行。

（2）输入合法的完整的记录，在完成之前放弃操作，检查表没有被更改。

（3）输入一个记录并故意漏掉一个数据项，检查表没有被更改。

（4）输入一个记录并故意有一个不合法数据项，检查表没有被更改。

（5）输入一个记录并使它的引用不存在，检查表没有被更改。

（6）事务中是否包含不确定的耗时操作，会导致并发失败、性能下降，如等待用户输入。

4）导入/导出测试

这时我们要关注输入输出设备不正确、繁忙、没有空间等情况下可能出现的问题；关注导入/导出文件类型不匹配，导入文件损坏或者内容不正确，当字符集表示方法不同时能否正确处理，以及数据恢复机制（尤其是系统升级的时候）可能出现的问题。

5）安全性测试

在对 C/S 应用软件测试时，我们要检查：在应用程序中，用户是否被正确锁定在访问路径和访问窗口中；在应用程序或者操作系统中，用户是否可能直接访问数据库文件；在数据库管理中，用户是否被赋予了不适当的权限；开发人员是否留了后门（这更多地依赖于代码审核和管理）；病毒检查；平台或者第三方系统本身的安全问题（如系统的已公布缺陷是否处理，是否打补丁了等）；请安全专家或黑客高手来对系统进行针对性检查或攻击。

6）Login/Logoff 测试

这里要对 C/S 应用软件检查：是否正确记录登录和退出日志；对于多次登录失败是否具有警告机制；口令强制修改措施是否能够正确执行；每次是否能够显示上次登录记录；空闲终端能否按要求退出（注意空闲条件判断，如屏幕保护程序）；是否符合规定的 License 要求。

7）日志测试

日志测试包括：是否正确记录日志内容，日志文件满、被删除、损坏、内容错误、访问权限错误是否能够正确处理，日志文件的安全和访问权限等。

3. C/S 系统的性能测试

对于 C/S 结构的系统来说，其承受大用户量并发访问的能力常常是用户着重考虑的一个

方面，最好的方法是使用测试工具来模拟多个用户端同时访问服务器，并使用性能监测工具获得关于服务器、数据库等用户关心的性能指标。

4. C/S 系统的测试步骤

与传统的软件测试一样，首先进行测试计划工作，但我们要格外注意多系统、多厂商的协调，要建立测试实验室，注意测试资源（尤其是软件/硬件资源）的配备和管理，要使用尽可能多样的系统组合，要关注性能测试（尤其关注 SQL，只有 20%的性能优化来自数据库管理），我们需要准备大量的数据（SQL 正确性需要小数据库，性能测试需要大数据库）；其次，进行测试设计和测试用例跟踪，特别要注意 C/S 应用软件的特殊错误类型和需要关注的测试点，而且必须考虑数据生成工具和性能测试工具的使用；再次，对测试中发现的问题或错误要进行缺陷报告和管理，特别要注意记录当时的系统/网络状态，注意记录当时的数据库和本机状态，注意缺陷的分离、重现和优化；最后，要对测试效果进行评估，与传统测试相比，要注意版本提交控制和配置管理。

14.3　B/S 系统测试

由于 B/S 架构具有前面叙述到的一些自身特点，使得基于 B/S 架构的软件测试与传统的软件测试既有相同之处，也有不同的地方。基于 B/S 架构的软件测试不但需要检查和验证应用是否按照设计的要求运行，而且要评价应用在不同用户的浏览器端的显示是否合适。

14.3.1　Web 应用测试

由于 B/S 应用程序一般要借助 IE 等浏览器来运行，而 Web 应用程序一般是 B/S 模式。

Web 应用程序，和用标准的程序语言，如 C、C++ 等编写出来的程序没有什么本质上的不同。然而 Web 应用程序又有自己独特的地方，就是它是基于 Web 的，而不是采用传统方法运行的。换句话说，它是典型的浏览器/服务器架构的产物。

所谓 Web 应用程序是一种可以通过 Web 访问的应用程序，它能够交付一组复杂的内容和功能给大量的终端用户。Web 应用程序的一个最大好处是用户能够很容易地访问应用程序，用户只需要有浏览器即可，不需要再安装其他软件。

一个 Web 应用程序是由完成特定任务的各种 Web 组件构成的并通过 Web 将服务展示给外界。在实际应用中，Web 应用程序是由多个 Servlet、JSP 页面、HTML 文件以及图像文件等组成。所有这些组件相互协调为用户提供一组完整的服务。

随着 Internet 的日益普及，现在基于 B/S 结构的大型 Web 应用越来越多，Web 应用软件已广泛应用于商业、工业、银行、财政、教育、政府和娱乐等领域。相对于传统的软件测试而言，基于 B/S 架构的 Web 测试有如下不同点：

（1）需要检查和验证是否按照设计的要求运行。

（2）还要测试系统在不同的用户的浏览器端的显示是否合适。

（3）重要的是还要从最终用户的角度进行安全性和可用性测试。

（4）Internet 和 Web 媒体的不可预见性使测试 Web 应用变得困难。

1. 功能测试

1）链接测试

链接是 Web 应用系统的一个主要特征，它是在页面之间切换和指导用户去一些不知道地址的页面的主要手段。链接测试可分为三个方面。首先，测试所有链接是否按指示的那样确

实链接到了该链接的页面；其次，测试所链接的页面是否存在；最后，保证 Web 应用系统上没有孤立的页面(所谓孤立页面是指没有链接指向该页面，只有知道正确的 URL 地址才能访问)。链接测试可以自动进行，现在已经有许多工具可以采用。链接测试必须在集成测试阶段完成，也就是说，在整个 Web 应用系统的所有页面开发完成之后进行链接测试。

2）表单测试

当用户给 Web 应用系统管理员提交信息时，就需要使用表单操作，例如用户注册、登录、信息提交等。在这种情况下，我们必须测试提交操作的完整性，以校验提交给服务器的信息的正确性。例如：用户填写的出生日期与职业是否恰当，填写的所属省份与所在城市是否匹配等。如果使用了默认值，还要检验默认值的正确性。如果表单只能接受指定的某些值，则也要进行测试。例如，只能接收某些字符，测试时可以跳过这些字符，看系统是否会报错。

3）数据校验

数据校验是指如果系统根据业务规则需要对用户输入进行校验，则需要保证这些校验功能能正常工作。例如，省份这个字段可以用一个有效列表进行校验。在这种情况下，需要验证列表完整而且保证程序正确调用了该列表(在列表中添加一个测试值，确定系统能够接受这个测试值)。在测试表单时，该项测试和表单测试可能会有一些重复。

4）Cookies 测试

Cookies 通常用来存储用户信息和用户在某应用系统的操作，当一个用户使用 Cookies 访问了某一个应用系统时，Web 服务器将发送关于用户的信息，把该信息以 Cookies 的形式存储在客户端计算机上，这可用来创建动态和自定义页面或者存储登录等信息。如果 Web 应用系统使用了 Cookies，就必须检查 Cookies 是否能正常工作。测试的内容可包括 Cookies 是否起作用，是否按预定的时间进行保存，刷新对 Cookies 有什么影响等。

5）设计语言测试

Web 设计语言版本的差异可以引起客户端或服务器端严重的问题，例如使用哪种版本的 HTML 等。当在分布式环境中开发时，开发人员都不在一起，这个问题就显得尤为重要。除了 HTML 的版本问题外，不同的脚本语言，例如 Java、JavaScript、ActiveX、VBScript 或 Perl 等也要进行验证。

6）数据库测试

在 Web 应用技术中，数据库起着重要的作用，数据库为 Web 应用系统的管理、运行、查询和实现用户对数据存储的请求等提供空间。在 Web 应用中，最常用的数据库类型是关系型数据库，可以使用 SQL 对信息进行处理。在使用了数据库的 Web 应用系统中，一般情况下，可能发生两种错误，分别是数据一致性错误和输出错误。数据一致性错误主要是由于用户提交的表单信息不正确而造成的，而输出错误主要是由于网络速度或程序设计问题等引起的，针对这两种情况，可分别进行测试。

7）应用程序特定的功能需求

除了以上基本的功能测试外，测试人员仍需要对应用程序特定的功能需求进行验证。例如某企业的订单管理应用软件，应尝试用户可能进行的所有操作：下订单、更改订单、取消订单、核对订单状态、在货物发送之前更改送货信息、在线支付等。

2. 性能测试

1）连接速度测试

用户连接到 Web 应用系统的速度根据上网方式的变化而变化，他们或许是电话拨号，或许是宽带上网。当下载一个程序时，用户可以等较长的时间，但如果仅仅访问一个页面就不会

这样。如果 Web 系统响应时间太长(例如超过 5 秒钟),用户就会因没有耐心等待而离开。另外,有些页面有超时的限制,如果响应速度太慢,用户可能还没来得及浏览内容,就需要重新登录了。而且,连接速度太慢,还可能引起数据丢失,使用户得不到真实的页面。

2) 负载测试

负载测试是为了测量 Web 系统在某一负载级别上的性能,以保证 Web 系统在需求范围内能正常工作。负载级别可以是某个时刻同时访问 Web 系统的用户数量,也可以是在线数据处理的数量。例如,Web 应用系统能允许多少个用户同时在线? 如果超过了这个数量,会出现什么现象? Web 应用系统能否处理大量用户对同一个页面的请求?

负载测试应该安排在 Web 系统发布以后,在实际的网络环境中进行测试。因为一个企业内部员工,特别是项目组人员总是有限的,而一个 Web 系统能同时处理的请求数量将远远超出这个限度,所以,只有放在 Internet 上,接受负载测试,其结果才是正确可信的。

3) 压力测试

进行压力测试是指实际破坏一个 Web 应用系统,测试系统的反映。压力测试是测试系统的限制和故障恢复能力,也就是测试 Web 应用系统会不会崩溃,在什么情况下会崩溃。黑客常常提供错误的数据负载,直到 Web 应用系统崩溃,接着当系统重新启动时获得访问权。压力测试的区域包括表单、登录和其他信息传输页面等。

负载/压力测试应关注以下问题。

(1) 验证系统能否在同一时间响应大量的用户:在用户传送大量数据的时候能否响应,系统能否长时间运行而不影响系统的性能。可访问性对用户来说是非常重要的。如果在用户使用系统时,总是得到“系统忙”的提示信息,那么他们可能放弃,并转向竞争对手;系统检测不仅要保证用户能够正常访问站点,而且在很多情况下,可能会有黑客试图通过发送大量数据包来攻击服务器;另外,出于安全的原因,测试人员应该知道当系统过载时,需要采取哪些措施,而不是简单地提升系统性能。

(2) 瞬间访问高峰。如果您的站点用于公布彩票的抽奖结果,最好使系统在中奖号码公布后的一段时间内能够响应上百万的请求。负载测试工具能够模拟多个用户同时访问测试站点。

(3) 每个用户传送大量数据。网上书店的多数用户可能只订购 1～5 本书,但是大学书店可能会订购 5000 本有关心理学介绍的课本。或者一个祖母为她的 50 个儿孙购买圣诞礼物(当然每个孩子都有自己的邮件地址),系统能处理单个用户的大量数据吗?

(4) 长时间的使用。如果站点用于处理鲜花订单,那么至少希望它在母亲节前的一周内能持续运行。如果站点提供基于 Web 的 E-mail 服务,那么站点最好能持续运行几个月,甚至几年。可能需要使用自动测试工具来完成这种类型的测试,因为很难通过手工完成这些测试。你可以想象组织 100 个人同时点击某个站点。但是同时组织 100 000 个人呢? 通常,测试工具在第二次使用的时候,它创造的效益,就足以支付成本。而且,测试工具安装完成之后,再次使用的时候,只要点击几下。

3. 可用性测试

1) 导航测试

导航描述了用户在一个页面内操作的方式,在不同的用户接口控制之间,例如按钮、对话框、列表和窗口等;或在不同的连接页面之间。通过考虑下列问题,可以决定一个 Web 应用系统是否易于导航:导航是否直观? Web 系统的主要部分是否可通过主页存取? Web 系统是否需要站点地图、搜索引擎或其他的导航帮助? 在一个页面上放太多的信息往往起到与预

期相反的效果。Web 应用系统的用户趋向于目的驱动,很快地扫描一个 Web 应用系统,看是否有满足自己需要的信息,如果没有,就会很快地离开。很少有用户愿意花时间去熟悉 Web 应用系统的结构,因此,Web 应用系统导航帮助要尽可能地准确。导航的另一个重要方面是 Web 应用系统的页面结构、导航、菜单、连接的风格是否一致。确保用户凭直觉就知道 Web 应用系统里面是否还有内容,内容在什么地方。Web 应用系统的层次一旦确定,就要着手测试用户导航功能,让最终用户参与这种测试,效果将更加明显。

2) 图形测试

在 Web 应用系统中,适当的图片和动画既能起到广告宣传的作用,又能起到美化页面的功能。一个 Web 应用系统的图形可以包括图片、动画、边框、颜色、字体、背景、按钮等。

图形测试的内容有:

(1) 要确保图形有明确的用途,图片或动画不要胡乱地堆在一起,以免浪费传输时间。Web 应用系统的图片尺寸要尽量地小,并且要能清楚地说明某件事情,一般都链接到某个具体的页面。

(2) 验证所有页面字体的风格是否一致。

(3) 背景颜色应该与字体颜色和前景颜色相搭配。

(4) 图片的大小和质量也是一个很重要的因素,一般采用 JPG 或 GIF 压缩。

3) 内容测试

内容测试用来检验 Web 应用系统提供信息的正确性、准确性和相关性。信息的正确性是指信息是可靠的还是误传的。例如,在商品价格列表中,错误的价格可能引起财政问题甚至导致法律纠纷。信息的准确性是指是否有语法或拼写错误。这种测试通常使用一些文字处理软件来进行,例如使用 Microsoft Word 的“拼音与语法检查”功能。信息的相关性是指是否在当前页面可以找到与当前浏览信息相关的信息列表或入口,也就是一般 Web 站点中的所谓“相关文章列表”。

4) 表格测试

表格测试需要验证表格是否设置正确。例如,用户是否需要向右滚动页面才能看见产品的价格;把价格放在左边,而把产品细节放在右边是否更有效;每一栏的宽度是否足够宽,表格里的文字是否都有折行;是否有因为某一格的内容太多,而将整行的内容拉长,等等。

5) 整体界面测试

整体界面是指整个 Web 应用系统的页面结构设计,是给用户的一个整体感。例如,当用户浏览 Web 应用系统时是否感到舒适,是否凭直觉就知道要找的信息在什么地方?整个 Web 应用系统的设计风格是否一致?对整体界面的测试过程,其实是一个对最终用户进行调查的过程。一般 Web 应用系统采取在主页上做一个调查问卷的形式,来得到最终用户的反馈信息。对所有的可用性测试来说,都需要有外部人员(与 Web 应用系统开发没有联系或联系很少的人员)的参与,最好是最终用户的参与。

4. 客户端兼容性测试

1) 平台测试

市场上有很多不同的操作系统类型,最常见的有 Windows、UNIX、Macintosh、Linux 等。Web 应用系统的最终用户究竟使用哪一种操作系统,取决于用户系统的配置。这样,就可能会发生兼容性问题,同一个应用可能在某些操作系统下能正常运行,但在另外的操作系统下可能会运行失败。因此,在 Web 系统发布之前,需要在各种操作系统下对 Web 系统进行兼容性测试。

2）浏览器测试

浏览器是 Web 客户端最核心的构件，来自不同厂商的浏览器对 Java、JavaScript、ActiveX、plug-ins 或不同的 HTML 规格有不同的支持。例如，ActiveX 是 Microsoft 的产品，是为 Internet Explorer 而设计的，JavaScript 是 Netscape 的产品，Java 是 Sun 的产品，等等。另外，框架和层次结构风格在不同的浏览器中也有不同的显示，甚至根本不显示。不同的浏览器对安全性和 Java 的设置也不一样。测试浏览器兼容性的一个方法是创建一个兼容性矩阵。在这个矩阵中，测试不同厂商、不同版本的浏览器对某些构件和设置的适应性。

3）分辨率测试

在分辨率测试中主要需要考虑以下问题：

（1）页面版式在 640×400、600×800 或 1024×768 的分辨率模式下是否显示正常？即在当前的分辨率模式下能否正常显示？

（2）字体是否太小以至于无法浏览？或者是太大？

（3）文本和图片是否对齐？

4）Modem/连接速率测试

是否有这种情况，用户使用 28.8 Modem 下载一个页面需要 10 分钟，但测试人员在测试的时候使用的是 T1 专线？用户在下载文章或演示的时候，可能会等待比较长的时间，但不会耐心等待首页的出现。最后，需要确认图片不会太大。

5）打印测试

用户可能会将网页打印下来。因此网页在设计的时候要考虑到打印问题，注意节约纸张和油墨。有不少用户喜欢阅读而不是盯着屏幕，因此需要验证网页打印是否正常。有时在屏幕上显示的图片和文本的对齐方式可能与打印出来的东西不一样。测试人员至少需要验证订单确认页面打印是正常的。

6）组合测试

最后需要进行组合测试。600×800 的分辨率在 MAC 机上可能不错，但是在 IBM 兼容机上却很难看。在 IBM 机器上使用 Netscape 能正常显示，但却无法使用 Lynx 来浏览。如果是内部使用的 Web 站点，测试可能会轻松一些。如果公司指定使用某个类型的浏览器，那么只需在该浏览器上进行测试。如果所有的人都使用 T1 专线，可能不需要测试下载时间（但需要注意的是，可能会有员工从家里拨号进入系统）。有些内部应用程序，开发部门可能在系统需求中声明不支持某些系统而只支持一些已设置的系统。但是，理想的情况是，系统能在所有机器上运行，这样就不会限制将来的发展和变动。

5. 安全性测试

Web 应用系统的安全性测试区域主要有：

（1）现在的 Web 应用系统基本采用先注册，后登录的方式。因此，必须测试有效和无效的用户名和密码，要注意到是否大小写敏感，可以试多少次的限制，是否可以不登录而直接浏览某个页面等。

（2）Web 应用系统是否有超时的限制，也就是说，用户登录后在一定时间内（例如 15 分钟）没有点击任何页面，是否需要重新登录才能正常使用。

（3）为了保证 Web 应用系统的安全性，日志文件是至关重要的。需要测试相关信息是否写进了日志文件、是否可追踪。

（4）当使用了安全套接字时，还要测试加密是否正确，检查信息的完整性。

（5）服务器端的脚本常常构成安全漏洞，这些漏洞又常常被黑客利用。所以，还要测试没

有经过授权，就不能在服务器端放置和编辑脚本的问题。

6. 接口测试

在很多情况下，Web 站点不是孤立的。Web 站点可能会与外部服务器通信，请求数据、验证数据或提交订单。

1）服务器接口

第一个需要测试的接口是浏览器与服务器的接口。测试人员提交事务，然后查看服务器记录，并验证在浏览器上看到的正好是服务器上发生的。测试人员还可以查询数据库，确认事务数据已正确保存。这种测试可以归到功能测试中的表单测试和数据校验测试中。

2）外部接口

有些 Web 系统有外部接口。例如，网上商店可能要实时验证信用卡数据以减少欺诈行为的发生。测试的时候，要使用 Web 接口发送一些事务数据，分别对有效信用卡、无效信用卡和被盗信用卡进行验证。如果商店只使用 Visa 卡和 Mastercard 卡，可以尝试使用 Discover 卡的数据（简单的客户端脚本能够在提交事务之前对代码进行识别，例如 3 表示 American Express，4 表示 Visa，5 表示 Mastercard，6 表示 Discover）。通常，测试人员需要确认软件能够处理外部服务器返回的所有可能的消息。

3）错误处理

最容易被测试人员忽略的地方是接口错误处理。通常我们试图确认系统能够处理所有错误，但却无法预期系统所有可能的错误。尝试在处理过程中中断事务，看看会发生什么情况？订单是否完成？尝试中断用户到服务器的网络连接。尝试中断 Web 服务器到信用卡验证服务器的连接。在这些情况下，系统能否正确处理这些错误？是否已对信用卡进行收费？如果用户自己中断事务处理，在订单已保存而用户没有返回网站确认的时候，需要由客户代表致电用户进行订单确认。

7. 故障恢复测试

故障恢复测试目的是确保系统能从各种意外数据损失或完整性破坏的各种软/硬件故障中恢复。所采取的方法是核实系统能够在 4 种状况下正确恢复到预期的已知状态：①客户/服务器断电；②网络通信中断；③异常关闭某个功能；④错误的操作顺序。

8. 安装/卸载测试

安装/卸载测试是测试软件在正常情况和异常情况下的安装/卸载状况。通常要核实这些行为：首次安装、升级、完整的或自定义的安装是否都能进行安装；磁盘空间不足、缺少目录创建权限等异常情况的安装。

基于 Web 的系统测试与传统的软件测试既有相同之处也有不同之处，它对软件测试提出了新的挑战。与传统的软件测试相比，基于 Web 的系统测试不但需要检查和验证系统是否按照设计的要求运行，而且要评价系统在不同用户的浏览器端的显示是否合适。更重要的是，还要从最终用户的角度进行安全性和可用性测试。

14.3.2 Web 应用性能测试方法

Web 应用的主要优点之一是：它允许多个用户同时访问系统资源；多个用户可以同时请求不同的服务，并获得不同特性的访问权。由于多用户支持是几乎每个应用取得最大成功的关键，因此必须评价系统在用户活动高峰期和正常时期执行重要功能的能力。

中国软件评测中心指出性能测试是软件测试的重中之重。性能测试包括三个方面：应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试。应用在

客户端性能测试的目的是考察客户端应用的性能，测试的入口是客户端。它主要包括并发性能测试、疲劳强度测试、大数据量测试和速度测试等，其中并发性能测试是重点。

1. 并发性能测试

并发性能测试的过程是一个负载测试和压力测试的过程，即逐渐增加负载，直到系统的瓶颈或者不能接收的性能点，通过综合分析交易执行指标和资源监控指标来确定系统并发性能的过程。

并发性能测试的目的主要体现在三个方面：①以真实的业务为依据，选择有代表性的、关键的业务操作设计测试案例，以评价系统的当前性能；②当扩展应用程序的功能或者新的应用程序将要被部署时，负载测试会帮助确定系统是否还能够处理期望的用户负载，以预测系统的未来性能；③通过模拟成百上千个用户，重复执行和运行测试，可以确认性能瓶颈并优化和调整应用，目的在于寻找到瓶颈问题。

当一家企业自己组织力量或委托软件公司代为开发一套应用系统的时候，尤其是以后在生产环境中实际使用起来，用户往往会产生疑问，这套系统能不能承受大量的并发用户同时访问？这类问题最常见于采用联机事务处理(OLTP)方式的数据库应用、Web 浏览和视频点播等系统。这种问题的解决要借助于科学的软件测试手段和先进的测试工具。

2. 疲劳强度测试

疲劳强度测试是采用系统稳定运行情况下能够支持的最大并发用户数，持续执行一段时间业务，通过综合分析交易执行指标和资源监控指标来确定系统处理最大工作量强度性能的过程。疲劳强度测试可以采用工具自动化的方式进行测试，也可以手工编写程序测试，其中后者占的比例较大。一般情况下以服务器能够正常稳定响应请求的最大并发用户数进行一定时间的疲劳测试，获取交易执行指标数据和系统资源监控数据。如出现错误导致测试不能成功执行，则及时调整测试指标，例如降低用户数、缩短测试周期等。还有一种情况的疲劳测试是对当前系统性能的评估，用系统正常业务情况下并发用户数为基础，进行一定时间的疲劳测试。

3. 大数据量测试

大数据量测试可以分为两种类型：①针对某些系统存储、传输、统计、查询等业务进行大数据量的独立数据量测试；②与压力性能测试、负载性能测试、疲劳性能测试相结合的综合数据量测试方案。大数据量测试的关键是测试数据的准备，可以依靠工具准备测试数据。

4. 速度测试

速度测试目前主要是针对有速度要求的关键业务进行手工测速度，可以在多次测试的基础上求平均值，可以和工具测得的相应时间等指标作对比分析。

14.4　SOA 应用软件测试

面向现代应用的系统测试首先要能够对具有复杂架构(特别是面向服务的 SOA 架构)的现代化复合应用进行测试。这是因为很多 IT 企业随着业务的逐步开放，他们共同面临着一些典型的问题。

(1) 信息孤岛比比皆是，信息共享实现困难。因为，这些企业大部分的业务系统是单独规划和建设的。这些系统的信息模型表示的实体如客户经常是相同的，但是代表这些实体的模型往往有很大的区别，这样就是成了不同系统的相关信息模型不能相互理解和共享。而且，不能通过接口调用的方式来访问和执行系统的功能，数据层面和组件层面都难于共享。

(2) 系统的开发不灵活,整体弹性小,市场竞争出击速度和响应速度慢。当业务需求发生变化时,对系统的修改和部署较难。由于牵一发而动全局,使响应周期长,风险过大,有时只能选择推倒重来。而当新系统建成后,与原系统互联互通时,往往又存在工作量过大、周期过长、成本高等各种问题。

为此,这些企业在系统升级与改造以及信息整合过程中,青睐于使用 SOA 架构,因为 SOA 是专门为解决如系统整合问题提出的 IT 理念。通过 SOA,我们可以:①协调服务,而不是协调大型的整体式应用程序(可以通过在多个应用程序中重复使用服务来降低开发成本,还可以通过在任意数量的应用程序间协调服务来提高现有企业资产的利用效率);②通过底层基础结构分离应用程序(使开发避开底层细节和复杂性,依赖松散耦合的自有服务来提高灵活性);③从面向技术转换为面向业务(SOA 允许随着业务的增长方便地添加、移除、合成/组合和重新合成服务,从而实现快速改变)。

14.4.1 基于 SOA 的 Web 服务

SOA 是一种 IT 体系结构,支持将企业的业务作为可重用业务任务进行集成,可在需要时通过网络访问这些服务和任务。SOA 提出了一种松散耦合的、基于标准的、面向服务的体系架构,以有效解决分布式、异构环境下,应用系统的集成问题。通过 SOA,可以逐步实现新的企业架构,使业务功能成为可重复使用的共享型标准服务。SOA 是声明型服务的集合,这些服务是独立的,相互之间为松散的耦合关系,但可通过策略进行控制。这些服务是自有的,被特别组合起来用于协调业务流程。SOA 是一种利用组件构造企业系统的方法,在这种方法中,系统是由一系列对应于更高级别的业务使用案例(其中每个案例包含通过网络提供的明确的功能集合)组成的。

1. SOA 与 Web 服务关系

SOA 是一种实施流程,拥有在应用程序间共享的服务。服务提供商创建服务,服务消费者利用这些服务。

合成服务由两个或两个以上服务组成,SOA 中使用的服务不限于 Web 服务,还可以包括其他技术,如分布式组件对象模型 (DCOM)和远程方法调用(RMI)XML 等。

Web 服务创建支持 SOA 的公用平台。Web 服务有多个组件:①简单对象访问协议(SOAP),为发送网络服务消息提供封装;②Web 服务定义语言(WSDL),它是构成 Web 服务的基础,服务提供商使用 WSDL 描述其服务;③可以搜索统一描述、发现和集成(UDDI)注册表,快速、方便和动态地查找和使用 Web 服务。

2. Web 服务概念

Web 服务是当今 SOA 最常用的技术,它是 SOA 部署的奠基石,它采用 SOA 体系架构,引入了一种新的 Web 应用的开发、部署和集成的模式,使业务应用程序可以在公司内部、客户、合作伙伴、供应商和其他相关实体之间共享各种通用传输协议、接口和事务模式。这不但改变了业务执行方式,产生了更快的服务部署,缩短了投资回报周期,还降低了总成本,并提高其适应性。SOA 提倡开放和重用的理念,正好是解决现今企业业务系统中信息孤岛现象的良方,是当今企业业务支撑系统实现整合的必经之路。

Web 服务是一组部署在应用服务器上的软件构件或软件组件,其服务接口及绑定形式可以通过 W3C 等国际标准组织制定的基于 XMI 的标准(如 WSDL,UDDI,SOAP 等)定义、描述、检索和调用,是实现各种异构平台上的应用间互连互通的主要技术方向,松散耦合性、简单性、高度可集成性、规范性、开放性和行业支持是 Web 服务的几大特点和优势。

Web 服务实现了“基于 Web 无缝集成”的目标，它是一种可透过网络存取、由多个应用程序组件组合所构筑的交互使用的环境。它描述了一种新出现的、重要的分布式计算范式，是利用 Internet 协议（如 HTTP 和 XML）实现远程调用的应用程序组件。Web 服务以其跨平台的可互操作性，得到了广泛的应用。而这些服务在我们应用之前，必须要经过测试检验，以确保它们在性能、质量、可靠性等方面满足应用要求。

14.4.2　SOA 应用测试

开展 SOA 应用开发，特别是向 SOA 转换面临着较大的风险。许多迁移的部分必须在连续的变化中紧密配合。各种服务具有不同的特性。随 SOA 而来的是多个利益相关方（如服务提供商和服务消费者），并且服务有独立的生命周期，与其开发和维护的方式相关。要成功实施 SOA，需要服务在面对不可避免的变化时，仍保持可互操作。

1. 对 SOA 应用测试的意义

SOA 架构加大了 IT 的复杂性，这些复杂性集中在需要管理的新关系上。由于服务依赖于基础设施，而应用程序取决于服务，因此存在技术关系。但是也存在组织关系。服务提供商需要进行跨角色和职责的协作来定义、开发和管理服务，消费者和提供商需要进行协作，议定服务级别协议 SLA 和其他有关使用服务的条款和条件。在服务的整个生命周期中，提供商必然要更改服务，客户的需求也必然会发生变化。因此，我们需要在服务的整个生命周期中解决持续出现的问题，例如：

（1）如何使服务满足功能需求；

（2）如何使服务在生产中扩展；

（3）如何管理测试几十甚至上百个服务的复杂性；

（4）如何快速确定和解决服务的响应时间问题；

（5）如何才能了解服务改变时所需测试范围的总体影响；

（6）如何才能使服务在整个企业中互操作，并且符合行业标准；

（7）如何才能在开发周期中尽早开始测试；

（8）如何才能在服务和系统的支持基础设施不存在或出现故障时对其进行测试；

（9）如何才能在服务发生变化和发展时在多个应用程序间共享服务。

需要对此复杂性的各个方面进行管理以改进 SOA 计划的结果。IT 部门需要具备将共享 SOA 抽象层作为自我实体测试的能力，而不管应用程序生命周期是否发生变化。通过测试 SOA 抽象层，IT 部门可以检验它是否继续随时间的变化为各种企业应用程序提供预期的功能和性能。

SOA 对测试还有其他多种意义，其中包括以下需求：

1）了解共享服务

我们需要对共享服务有广泛的了解，使不同的部门和个人能在质量流程中发挥重要作用。其中一个最大的难题是通过了解更广泛的业务层和共享服务，了解业务影响和与部署相关的问题的正确优先级。

2）了解共享服务的弱点

服务可以由不同部门提供，共享服务可以由多个应用程序利用。在服务的整个生命周期中，要确认服务中的更改不会损害其他利用服务的应用程序，必须进行测试。服务提供商可以更改为特定应用程序开发的服务，而无须了解这些更改会对共享这些服务的其他应用程序产生的影响。共享服务包括安全性和可靠的消息传递；可以按照策略对它们进行管理；对于任

何给定的服务，都存在许多潜在的故障点。因此，需要对服务进行连续的测试，确保它们不会造成瓶颈或者在其整个生命周期中无法按预期执行。

3）管理持续质量

服务需要涵盖质保重点，IT 部门需要建立和管理用于管理持续质量的 QA 流程。质量目标必须通过集中的质保和分布式任务认真定义，以便优化支持服务中的 IT 资源使用。

4）管理大量服务和数据

合成应用程序依赖于可在整个企业中使用的共享服务。QA 可能需要管理成百甚至上千种经常更改和发展的共享服务。

5）管理新的 SOA 测试方面

因为要使用网络服务描述语言 WSDL 描述网络服务，所以需要开发公用标准，使不同的应用程序可以依赖共享服务。例如，如果架构师定义了内部标准，那么，开发人员就需要根据这些标准编程。但是，不同的部门可能使用了不同的编程工具包，因此必须进行互操作性测试。开发组织与以往的管理相比，对互操作的需要增加了迁移部分的数量，减少了控制。

服务必须符合组织标准，使其可以由整个企业中的不同 QA 团队进行连续测试；组织需要拥有验证服务是否符合已立标准的能力。QA 还要能执行边界测试，使用符合 WSDL 规范的数据和元数据调用每项服务操作，例如，QA 可能需要拥有测试 SOAP 标题边界的能力。它们还需要拥有测试服务性能以及在不同的阈值下模拟性能的能力，确保服务随时间扩展。

6）使服务可用于测试

服务需要在开发周期的早期就可用于测试。就 WSDL 服务描述达成共识后，QA 就可以在服务开发完成前开始实施实际测试了。我们还需要拥有测试缺少服务后台的复杂环境的能力。例如，可以对依赖访问敏感数据（如员工社保号）的服务进行测试，而不必使敏感数据在整个 QA 周期中自由访问。

7）创建生命周期质量流程

在服务的整个生命周期中都必须测试服务，包括从开发阶段到生产部署，直到生命结束。测试应集成到生命周期流程中，使服务可以随时间发展，同时确保服务能与多个应用程序成功配合。

8）分析变更的影响

QA 面临的主要难题之一是确定如何有效地测试发生的变更。我们需要确定哪些变更可能会引起大的风险，哪些是 QA 投入精力的最佳位置，以及在哪些位置可以执行最少的验证来优化资源。在传统的应用程序测试流程中，QA 面临的难题是了解开发所做的变更，决定是否需要对其进行测试。考虑到涉及的服务的数量、内部依赖关系的数量以及使用共享服务的应用程序的数量，了解变更的影响在共享服务中有更重要的意义。

9）实施功能和性能测试

需要在服务的整个生命周期中测试服务，确保它们按承诺发挥作用，并提供预期的性能和结果。SOA 还蕴涵了其他功能和性能测试难题。在针对企业应用程序的传统测试中，测试工程师能够针对图形用户界面（GUI）进行测试。通过 GUI 客户端，可以更方便地了解业务流程和数据流，但是，通过 SOA，需要根据 WSDL 测试服务，不提供 GUI 来简化业务流程和数据流。

10）执行异步测试

通过 SOA，服务通常不会同步，因此，必要时，QA 必须能不按顺序测试业务流程。QA 需要拥有解决异步通信机制的能力，这些机制包括轮询数据、通过代理轮询和通过 Web 服务寻

址及其他标准回调等。

11）测试能公开服务的其他技术

我们还需要拥有能支持其他具备公开服务功能的运行测试的能力，这些服务包括 Java 消息服务（JMS），它是在两个或多个客户端间发送消息的中间件应用程序编程接口（API），以及 RMI，它是执行远程程序调用的 API。

2. SOA 应用的测试难点

面向服务软件架构是一种新型的软件架构思想，同面向对象思想的出现一样，带来了对软件测试的新的挑战。SOA 在更高的层次上实现了软件重用和封装，通过业务流程的方式将原来独立的软件应用系统结合在一起共同完成一系列的功能，这极大地方便了企业信息整合。但也给最终的 SOA 应用测试带来了极大的难度。

基于 SOA 架构开发的应用系统采取的测试方法普遍还是传统的面向对象的测试方法，由于系统架构的改变，测试方法也需要变化。在对基于 SOA 的应用系统进行测试时需要考虑三个问题：①源代码不可见；②程序自身的问题（对于多个相互独立的 Web 服务分布单元，需要处理其间的通信、同步以及并发操作）；③分布运行环境的问题（包括程序与运行环境的合作、网络通信的完整性、平台无关性以及异构系统间的互操作性）。

基于 SOA 的应用系统是由封装好的可以提供服务的构件组合而成，每一个服务构件都有一系列定义好的接口，服务构件之间通过这些接口互相提供服务完成应用系统的功能。通过测试服务构件提供的服务以及服务构件在提供服务过程中状态的变化，可以验证这些服务构件之间交互正确与否。

基于 SOA 的软件测试的难点主要包括服务测试、业务流程测试、基础设施测试和系统性能测试这四个方面。

1）服务测试

服务测试是最重要的，因为核心服务是 SOA 的基础。然而，各种服务在编写方面彼此之间差别很大，因为它们的开发者是不同的，有些服务粒度也许比较粗，而有些服务可能会很细，还有一些服务则可能设计粗劣。还有些服务也许会建立在现有界面和 API 上，因此它们就更加复杂，更需要进行质量保证试验，因为需要在一个中间层之外再加一个中间层。这其中并没有什么技巧，主要就是验证各项服务的用途、界面功能是否正确以及验证 WSDL 和规划等内容。

服务测试相当于单元测试，在这个阶段，需要对服务接口、局部数据结构的完整性、边界条件、覆盖条件和出错处理等进行测试。

2）业务流程测试

除了服务测试之外，我们还必须测试服务被加入到业务流程中或混合应用中的整个应用情况。事实上，业务流程测试相当于集成测试/系统测试。在这个阶段，可以根据业务流程的特点，观察系统级的输入和输出是否符合预期。

3）基础设施测试

SOA 的实施除了需要具体的应用软件系统提供的服务外，还需要一系列的基础设施才能正常运行，这些基础设施一般包括服务注册和发现系统（一般为 UDDI）、授权系统等。这些系统也是以独立的接口提供服务，需要进行服务测试。由于这些系统只有在业务流程中才能真正发挥作用，也需要将其放在业务流程中进行测试。

基础设施类似于一般软件应用系统中调用的 API、SDK。但是，由于 SOA 的应用针对性很强，不同的 SOA 架构基础设施不可能完全一样，如授权系统，由于涉及的系统不同，不同的

SOA 之间肯定存在差异,所以在测试时不能假定基础设施的正确性,必须进行测试。

4) 系统性能测试

性能测试也很重要,SOA 是一个松散耦合的系统架构,其中业务流程通过调用不同软件应用系统提供的服务完成,这些应用系统本身的性能可能因为设计或被分配计算、处理任务的多少而差别很大,一个应用系统的处理速度慢会造成整个 SOA 系统架构的瓶颈。SOA 的大部分质量问题都跟性能有关。SOA 中的性能测试就是对服务、构成、业务流程和系统等不同级别的测试问题。在测试系统的整体性能时,必须沿着体系结构对数据流图中最高层到最底层进行分解,找出系统中存在问题的组件。

SOA 中的服务可能封装了普通应用软件系统中的一系列处理过程,如移动提供的短信发送服务就可能封装了用户认证、欠费查询、短信发送、计费等多个模块,而业务流程由于集成了大量服务,性能方面可能暴露的问题更大。有实验数据表明,SOA 的性能在目前情况下可能只相当于不采用 SOA 时的 10%,所以性能测试对于 SOA 应用的意义超过了以前任何一种软件测试。通过性能测试,可以找到瓶颈服务从而对其进行调优,也可以根据各个组成服务的性能特点在保证处理逻辑正确的情况下对业务流程进行调优,这些都会带来 SOA 的性能的极大提高。

3. 基于 SOA 的测试技术

同上,基于 SOA 的软件测试内容主要包括服务测试、业务流程测试、基础设施测试和系统性能测试这四个方面。其中对于服务测试主要集中在功能测试上。

1) 服务的功能性测试

功能测试是对基于 SOA 的应用系统的功能进行测试,主要是检验服务交互时可能引发的消息错误,即看调用过程中是否有无效的操作,以及服务是否可重复调用的问题。

SOA 中的业务通过组合多个应用软件系统提供的服务而形成,从 SOA 的范畴来看,服务是组成 SOA 业务流程的有机模块,对服务的测试相当于单元测试,但是从服务对应的应用系统来说,一个服务可能通过调用应用的多个模块而完成了一系列的功能,对服务的测试除了对服务本身输入输出的正确性(黑盒)进行测试外,还应该包括更深层次对服务所涉及的应用软件系统模块进行测试,在这个意义上,对 SOA 中具体服务的测试实际上是对提供服务的应用软件系统与该服务相关的部分进行集成测试。

2) 业务流程分析和测试

业务流程是 SOA 实现具体功能的有机单元,对其的测试可以采用类似传统软件测试中的数据流分析和测试的方法。一个业务流程中可能存在多个分支,测试需要保证每个分支都能被充分测试到。

3) 基础架构测试

由于基础架构是 SOA 得以运行的必备组件,这一块的测试必须针对每一个具体实施来进行。不存在移植后就不需测试的侥幸。比如授权系统,当每一个新的应用系统加入这个架构,都必须进行重新测试,即使这个新的应用系统采用的是同原有系统中相同的认证方式,因为对它的测试不仅是测试服务和业务流程本身,也是对新系统提供的服务接口的测试。

对基础架构的测试也可分为服务测试和业务流程测试两块。前者测试基础架构作为一个单独系统的功能完整性,后者则是业务流程中必不可少的一个测试条件。

4) 性能测试

性能是 SOA 应用的一大障碍,SOA 提供了灵活的业务组合机制以及通用的服务调用方式(如 Web 服务),但是带来的牺牲之一便是性能的下降。SOA 性能测试的目的性很强,一是

发现性能瓶颈,并对瓶颈部分进行调优;二是对业务流程的逻辑结构进行调优,从而提高整体性能。从性能测试来说可包括以下几个方面:

(1) 服务性能测试。通过对单个服务的压力测试,可以知道此服务的最大负载、响应时间等参数,从而作为以后判断瓶颈的依据。

(2) 网络流量测试。由于 SOA 是针对具体的环境进行实施,要能预测实施地点的实际网络带宽,通过测试业务相互调用的流量测试,尽量将大流量的调用放在带宽较大的地点,而对于较远程的调用则尽量采用各种减小流量的方法,如数据库采用存储过程、XML 采用压缩等方法。

(3) 业务流程性能测试。通过测试业务流程的单位时间执行次数,并根据其中业务逻辑和对服务性能测试的结果判断出瓶颈,如果该瓶颈所在的服务可以调优则对其进行优化,如果由于某些原因(业务系统不能变动或是开发人员的离开)无法对单个服务进行优化,则需要对业务的逻辑进行优化,如通过任务的调度,将该瓶颈所在的服务从关键路径中调离转为并发(如果可以的话)。

(4) 服务器、服务器软件等外围条件测试。服务的发布和调用一般由专用的服务器软件完成,如 Web 服务可以通过 IBM 公司的 WebSphere、BEA 公司的 Web Logic 或是开源的 Tomcat 来发布,这些服务器软件的执行性能各不相同,在选用时需要对其进行一些测试,了解其性能中的并发数、响应时间等具体参数,结合需要实施的 SOA 的具体情况进行选择。同时服务器的性能也是一个重要指标。往往瓶颈的出现不是因为软件的问题而是应用服务器出了问题。

性能测试主要采用基准测试、容量测试和 soak 测试这三种方法。

(1) 基准测试是用来确定被测应用程序是否存在性能衰退,并且收集可重复性能测试结果以作为性能基准。基准测试的最好方法是每次测试只改变一个参数,包括响应时间驱动的测试和吞吐量驱动的测试。

(2) 容量测试是看被测应用程序在一定测试环境下能够达到的最大处理能力。容量测试尝试模拟更加接近真实用户使用的环境,并且用更为真实的用户负载来测试 SOA 平台的容量规模。

(3) soak 测试是在用来测试基于 SOA 平台程序的健壮性,是在一个稳定的并发用户上进行的 long-run 测试,通过 soak 测试能够发现内存泄露等严重的问题。

14.4.3　Web 服务测试

Web 服务作为实现 SOA 的一种形式,已得到广泛的关注与应用。对 Web 服务进行充分测试可保证 Web 服务的质量,然而由于 Web 服务所具有的特点,传统的软件测试技术方法不再适用于基于 SOA 架构的应用系统的测试,所以对基于 SOA 架构的 Web 服务测试难度加大。

1. Web 服务测试的难点

Web 服务处于分布式计算的核心位置,其分布应用、具有各种运行时行为、涉及多种标准协议,可能在硬件、软件、通信、对象管理等各个环节出现各种缺陷。其体系结构和应用的复杂性,以及技术和规范不断地发生变化,对测试研究提出了新的挑战。Web 服务的测试特点与难点主要体现在以下几个方面。

(1) Web 服务的开发环境与其应用环境有很大的不同。在发布之前,很难对其实际的运行场景进行预测,如访问的用户类型、并发用户数量、Web 服务调用的装载模式和访问方式等,这些差异和应用的不确定性都增加了 Web 服务测试的困难。

制和故障恢复能力，也就是测试 Web 服务系统会不会崩溃，在什么情况下会崩溃。

并发性能测试的目的主要体现在三个方面：①以真实的业务为依据，选择有代表性的、关键的业务操作设计测试案例，以评价系统的当前性能（当扩展应用程序的功能或者新的应用程序将要被部署时，负载测试会帮助确定系统是否还能够处理期望的用户负载，以预测系统的未来性能）；②通过模拟成百上千个用户，重复执行和运行测试，可以确认性能瓶颈并优化和调整应用，目的在于寻找到瓶颈问题；③负载测试，它是采用系统稳定运行情况下能够支持的最大并发用户数，持续执行一段时间业务，通过综合分析交易执行指标和资源监控指标来确定系统处理最大工作量强度性能的过程（如出现错误导致测试不能成功执行，则及时调整测试指标，例如降低用户数、缩短测试周期等）。

2）应用在网络上性能的测试

应用在网络上性能的测试重点是利用成熟先进的自动化技术进行网络应用性能监控、网络应用性能分析和网络预测。网络应用性能分析的目的是准确展示网络带宽、延迟、负载和 TCP 端口的变化是如何影响用户的响应时间的。

网络性能测试可以解决多种问题：①服务请求端是否对数据库服务器运行了不必要的请求；②当服务提供者从客户端接受了一个请求服务，应用服务器是否花费了不可接受的时间请求 Web 服务；③在系统试运行之后，需要及时准确地了解网络上正在发生什么事情，什么应用在运行，如何运行；④多少 PC 正在访问 Web 服务；⑤哪些应用程序导致系统瓶颈或资源竞争。

网络应用性能监控以及网络资源管理对系统的正常稳定运行是非常关键的。在大多数情况下用户较关心的问题还有：哪些应用程序占用大量带宽，哪些用户产生了最大的网络流量。考虑到系统未来发展的扩展性，预测网络流量的变化、网络结构的变化对用户系统的影响非常重要，根据规划数据进行预测并及时提供网络性能预测数据。

3）应用在 Web 服务提供端性能的测试

对于应用在服务提供者端上性能的测试，可以采用工具监控，也可以使用系统本身的监控命令。

3. Web 服务测试的基本方法

由于 Web 服务涉及服务提供者、服务中介者和服务客户端，因此他们都需要参与到测试中来。服务提供者拥有服务实现的源代码，可以进行“黑盒”和“白盒”的充分测试。但是在服务提供者方进行的测试不能真实反映实际的运行情况，比如网络负载、客户端的数量等。因此需要在服务中介者和服务客户端进行进一步的测试，特别是在服务客户端。服务客户端是 Web 服务的直接使用者，在客户端进行测试对整个 Web 服务质量评估是非常重要的。因此测试也主要基于客户端进行。Web 服务测试的基本方法有功能测试、负载测试、压力测试、回归测试以及服务监测。

4. Web 服务测试的整体结构

根据 Web 服务架构和业务模型，Web 服务测试的整体结构可分为三个层次（基础设施测试、独立的 Web 服务测试以及 Web 服务集成测试）以及对 Web 服务测试的组织与管理。

1）Web 服务基础设施的验证与确认

虽然标准化和开放性是 Web 服务的主要宗旨，但与传统的应用系统相比，Web 服务中间件的稳定性和可靠性相对薄弱。一方面，是由于 Web 服务的整个技术架构还尚未成为成熟的产业标准。在 W3C 联盟以及相关研究机构的大力支持下，标准规范体系还处于不断发展和完善的过程中，不同的标准之间的概念模型和表示体系可能互不兼容，并进一步导致相关应用

程序的不兼容。

另一方面，Web 服务规范体系采用 XML 作为基本的编码语言。由于 XML 技术简单、灵活、可伸缩、易定制，基于 XML 数据描述和 XML Schema 数据建模定义的 Web 服务的协议栈具有标准化和开放性的特点。但是同时，XML 为描述性语言，其通用性和灵活性为协议的可证明性提出了挑战。

当 Web 服务用于高可信性或实时要求较高的应用中，需要更严格的验证方法。例如，SOAP RPC（Remote Procedure Call）是 Web 服务的基础和核心协议，由于编码格式的不同，SOAP RPC 与传统的 DCE RPC 及 ONC RPC 采用了完全不同的消息映射和消息处理机制。SOAP 还处于完善过程中，对于 SOAP 本身的验证还有待于进一步加强。

2）独立的 Web 服务测试

Web 服务首先作为独立的功能节点发布，再通过工作流定义和解析动态集成为完整的业务流程。独立的 Web 服务测试就是从以下三个方面保证各服务节点的质量：

（1）服务的实现应在功能、性能等各方面与发布的服务描述相一致。为验证一致性，除服务提供者外，服务中介及用户都应能在一定的安全约束下，远程测试该服务。

（2）由于服务发布的开放性，对于每一个服务请求，可能存在多个满足需求的服务描述。服务中介应能根据一定的度量和评价标准，对多个服务进行测试、比较和评估，并依照需求的满足程度排序。

（3）在服务实现的演化过程中，应建议一定的机制来支持对不同版本的跟踪及回归测试。

3）集成的 Web 服务测试

通过对服务流描述的解析，Web 服务可以动态地集成。Web 服务集成的描述、解析和执行将是 Web 服务区别于其他分布式计算技术的一个主要特征。目前已经提出了多种 Web 服务描述语言，如 IBM 的 WSFL、微软的 XIANG 等。

集成的 Web 服务测试就是在服务流描述执行前，通过静态验证以及动态模拟的方法，确认服务描述能够正确地描述业务需求，能够由服务中介正确地解析，并能由所有服务节点正确地执行。

4）Web 服务测试组织和管理

传统的应用系统开发中，测试过程和测试资产通常为集中管理和监控的模式。在传统的测试流程中，测试架构人员根据系统特征点，设计测试用例和测试场景。测试工程师执行测试，捕捉测试结果，报告缺陷，并跟踪缺陷的修复过程。对于修复的系统，还要通过回归测试来确认缺陷已经被正确地更改，并且系统没有引入新的缺陷。

Web 服务的分布式特征使这一过程变得更加复杂。服务提供者、中介和使用者三方需要在一个分布合作的环境下，共同建立和维护测试方案、测试结果和缺陷数据库，并跟踪缺陷修复过程。测试及回归测试都必须通过自动化的测试引擎，通过互联网远程调度、执行。为模拟用户的实际应用环境和场景，亦需在一个分布式的环境下，统一调度多个测试引擎。不同测试方案、测试引擎和测试运行的测试结果需要能够进行综合和分析，并最终给出评价结果。

习题

1. B/S 应用与 C/S 应用相比各有什么特点，基于 C/S 架构和 B/S 架构的应用测试与传统的软件测试有什么不同之处？

2. C/S 系统及 B/S 系统（Web 应用）测试包括哪些测试内容？怎样进行这些内容的

测试?

3. 如何进行 C/S 系统及 B/S 系统(Web 应用)的性能测试?
4. Web 应用都有哪些测试模型?
5. Web 应用测试有哪些测试内容,怎样进行这些内容的测试?
6. 什么是 SOA 架构? SOA 与 Web 服务是什么关系? Web 服务有什么特点?
7. SOA 测试包括哪些测试内容? 涉及哪些测试技术?
8. Web 服务测试包括哪些测试内容? 如何测试?



第15章 移动应用软件测试

移动互联网的发展深刻地改变了人们的生活方式。从数量来看，最近一年的智能手机出货量已经超过了PC，人们的生活更多地开始依赖智能终端。在智能应用得到广泛发展以后，智能终端的安全问题也就随之出现。

我国3G发展已经进入了规模发展阶段，使用3G的用户相当普遍。3G用户的规模发展带动了智能机的迅速普及。随着智能终端的普及，手机安全问题也日益凸显。

一方面，互联网上原有的恶意程序传播、远程控制、网络攻击等传统网络安全威胁向移动互联网快速蔓延，导致智能终端面临着安全威胁；另一方面，智能终端和用户个人利益关系更加密切，恶意吸费、用户信息窃取、诱骗欺诈也随之出现。智能手机面临着前所未有的安全危机。

目前，智能手机应用模式是：把智能终端作为信宿，移动应用软件作为信息载体，应用商店作为流通渠道，应用服务器作为信息源。基于该模式，它的4个层面都存在着不同程度的安全问题。因此，国内外有关机构及企业已将解决智能终端的安全问题提上了议事日程。他们针对移动应用软件，开展测试工具的开发和测试平台的搭建，以期对移动应用软件进行全面测试，如功能测试、安全性测试、性能测试、易用性测试、终端适配测试等，解决移动软件所面临的质量问题及安全危机。

15.1 移动应用测试的困难

自2010年开始，人们开始热衷于通过App使用移动互联网，移动应用App已经渗透每个人的生活、娱乐、学习、工作当中，并使得各种信息变得无处不在，世界融合的趋势大大增强。另外，令人激动、兴奋且具有创造性的各种App犹如雨后春笋般交付到用户手中。各类智能终端也在快速发布，而开发者对于全球移动设备/智能终端的质量和性能却掌握甚少，App与设备的兼容性问题常常导致用户投诉，令开发者十分沮丧，App测试与服务质量保证矛盾十分突出。

移动开发和测试的一个重要难题，就是应用在开发和测试过程中，必须使用手机/终端真实环境进行系统测试，才有可能进入商用。由于手机/终端操作系统的不同，以及操作系统版本之间的差异，使得真机系统测试这个过程尤其复杂，涉及终端、人员、工具、时间、管理等方面的问题。

1. 手机/终端配置测试实验室

首先必须购买足够多的手机/终端，包括不同操作系统、不同版本、不同分辨率，甚至不同厂商，目前市场上的手机/终端平台有iOS、Android、Symbian、WP、Blackberry、Linux等（集中度较高的是iOS、Android和WP系统），平台之间存在较大差异，语言和标准完全不同。以

力、缺乏文档的代码、缺乏 App 开发工具及 SDK 和人员的疏忽等原因引发的错误，通过测试能够发现、找出其中的错误，解决错误，从而提高 App 的质量。

1. App 单元测试

App 单元测试是对 App 的基本组成单元来进行正确性检验。集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能。

目的是检测 App 模块对 App 产品设计说明书的符合程度。

测试类型一般是“白盒”测试，测试范围为单元内部的数据结构、逻辑控制、异常处理。

评估标准是逻辑覆盖率。

2. App 集成测试

App 集成测试是测试模块或子系统组装后功能以及模块间接口是否正确，把已测试过的模块组装起来，主要对与设计相关的 App 体系结构的构造进行测试。

目的是检测 App 模块对 App 产品概要设计说明书的符合程度。

测试类型一般是“灰盒”测试，测试范围为模块之间接口与接口数据传递的关系，以及模块组合后的功能。

评估标准是接口覆盖率。

3. App 系统测试

App 系统测试是将已经确认的 App 程序、移动终端、外设、网络等其他元素结合在一起，进行信息系统的各种组装测试和确认测试，系统测试是针对整个产品系统进行的测试，目的是验证系统是否满足需求规格说明的定义，找出与需求规格说明不符或与之矛盾的地方，从而提出更加完善的方案。App 系统测试发现问题之后要经过调试找出错误原因和位置，然后进行改正。

App 系统测试是基于系统整体需求说明书的“黑盒”测试，应覆盖系统所有的部件。对象不仅仅包括需测试的 App 软件，还要包含 App 软件所依赖的硬件、外设甚至包括某些数据、某些支持软件及其接口等，基于本地及不同地区、网络等真实终端，测试、检查已实现的 App 是否满足需求规格说明中确定了的各种需求，以及 App 配置是否完全正确。

App 系统测试的评估标准是测试用例对需求规格说明的覆盖率。

系统测试过程参见图 6-27。系统测试各阶段的任务参见表 6-6。

移动 App 系统测试的终端如图 15-1 所示。

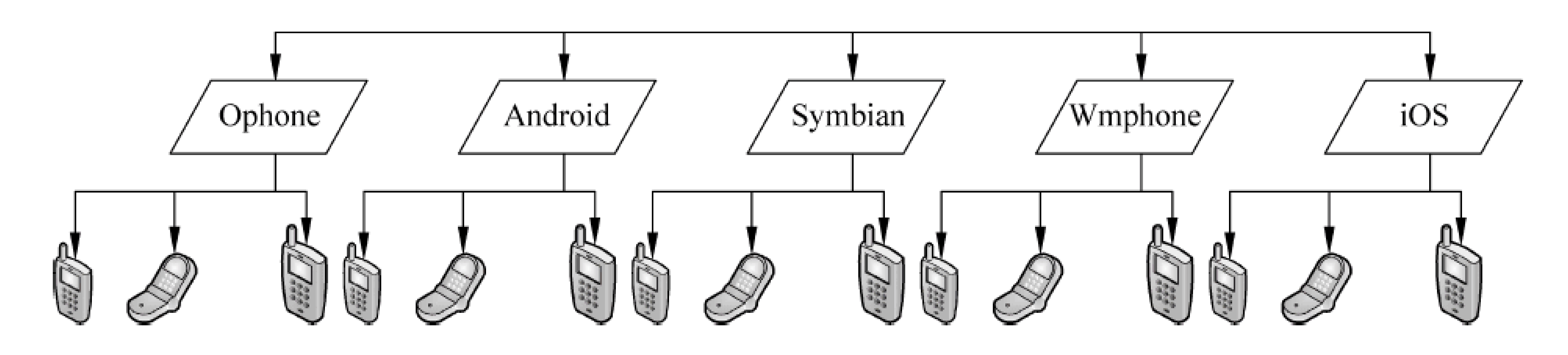


图 15-1 移动 App 的测试终端

15.2.2 移动 App 测试类型

目前主流的 iOS、Android 和 WP 等 OS 以及各平台，都相应地提供了不同程度的单元、集成测试工具，可以在模拟器、沙箱环境下进行“白盒”、“灰盒”测试以及调试。

但 App 存在着大量的软硬件交互，而这些都需要在真实的终端上通过“黑盒”测试方法进行系统测试，需要将经过集成测试的软件，作为移动终端的一个部分，与系统中其他部分结合起来，在实际运行环境下对移动终端系统进行一系列严格有效的测试，以发现软件潜在的问

题，保证系统的正常运行，验证最终软件系统是否满足用户规定的需求。

然而，由于 OS 版本、硬件异常迅猛的发展速度，平台始终没有有效地提供符合 App“黑盒”系统测试的工具与方法，大量的移动 App 测试还停留在纯人工状态，效率十分低下。终端、版本的碎片化，更加剧了这一问题的严重性。

自己开发，或借助第三方工具、平台，进行自动化的移动互联网 App 系统“黑盒”测试，是提升效率和测试质量的有效方法。

移动互联网是极快速发展的新兴产业，没有成功经验可循，只有市场和用户才是检验产品是否好坏的终极标准。借助传统软件测试方法和规律，可以有效地提升 App 的程序质量和用户体验。

1. 冒烟测试

冒烟测试(Smoke Testing)的对象是每一个新编译的需要正式测试的 App 版本，目的是确认软件基本功能正常，可进行后续的正式测试工作。冒烟测试的执行者是版本编译人员。

App 程序在编写开发过程中，内部需要多个版本，但是只有有限的几个版本需要执行正式测试(根据项目开发计划)，这些需要执行的中间测试版本，在刚刚编译出来后，开发人员需要进行基本性能确认测试，验证 App 是否能正确安装、卸载，以及操作过程和操作前后对系统资源的使用情况，针对终端硬件及 ROM 版本的各维度，与 App 安装、卸载不适配情况、隐患原因分析报告，最终确认是否可以正确安装/卸载，主要功能是否实现，是否存在严重死机、意外崩溃等 Bug。

如果通过了该测试，则可以根据正式测试文档进行正式测试。否则，就需要重新编译，再次执行确认测试，直到成功。

如果发现问题，就要有效地发现导致问题出现的原因。例如在 Android App 测试中，某些终端有时会出现应用程序错误需要强行关闭的提示，但又找不到重现这个问题的步骤，这个是 App 的问题还是系统的问题呢，应该怎么判断呢？这通常需要有 Log 日志才可以判断。Android App 出现 Crash 的情况，一般有两方面的原因，如果 Log 日志中出现 System_server，则为系统问题；如果 Log 中出现 Shutdown VM，则代表应用程序的问题；还有一种情况是出现 Died，这个是进程死掉导致，包含系统主动杀死的情况。

2. 功能测试

由于企业正在寻求改善用户体验并更快地为市场带来变化，QA 团队不仅必须要验证移动应用的功能，同时还要使现有的测试流程和方法适应基于移动设备/智能终端的执行方式，确保跨多种环境的一致性表现，并更快取得测试结果。

这些移动应用复杂性和多变性所带来的挑战使得一切变得更加困难重重。人工测试太过烦琐缓慢，无法满足移动用户的需求。测试团队还将认识到，如果创建的测试方案无法跨多个设备、运营商、操作系统和地理位置使用，那么将是不可接受的。为了支持敏捷性的整体业务目标，测试者必须更新其自动化操作，使其涵盖移动设备/智能终端；除此之外，根本没有其他方法能足够快地跨所有测试组合获得测试结果。如果我们建立的一整套测试方案相对而言更易于维护，那么随着应用的频繁发布，测试方法的重用率(以及投资回报)将非常高。

其他需要回答的问题还有，如何评估基于仿真器测试的解决方案以及真实设备测试解决方案的风险和预算，以及相应的需求。尽管大多数团队认为仿真器测试已经足够，但当我们的用户在他们的个人设备上使用应用后，可能会给出不同意见。问题是：对我们的业务而言，风险是什么？我们是否需要对真实设备测试投入更多来保证良好体验？我们是否需要在全球部署设备以测试基于位置的服务？

最后也很重要的一点是，大部分移动应用无法独立存在：它们是更大系统的接入点。团队想要降低风险并提高测试用例对这些综合性应用的覆盖率，必须不仅能够测试移动应用的功能，还需要测试事务处理功能，因为它贯穿了综合性系统的不同组件、GUI、服务和数据库。这确实是一项挑战，因为众多测试团队目前尚未能充分解决综合性应用的问题。

因此，我们说功能测试是移动App测试最关键的环节，根据产品的需求规格说明书和测试需求列表，验证产品的功能实现是否符合产品需求规格说明。

功能测试的目标主要包括：①是否有遗漏需求；②是否正确地实现所有功能；③隐含需求在系统是否实现；④输入、输出是否正确。

移动App的功能测试应侧重于所有可直接追踪到用例、或业务功能和业务规则的测试需求。这种测试的目标是核实数据的接受、处理和检索是否正确，以及业务规则的实施是否恰当。

功能测试基于“黑盒”技术，通过图形用户界面(GUI)与应用程序进行交互，并对交互的输出或结果进行分析，以此来核实应用程序及其内部进程。

3. 图形用户界面测试

图形用户界面(GUI)测试用于核实用户与App之间的交互，包括用户友好性、人性化测试。

一个好的App要有一个极佳的分辨率，而在其他分辨率下也都可以运行。GUI测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。另外，GUI测试还可确保GUI中的对象按照预期的方式运行，并符合公司或行业的标准。

GUI测试主要测试在不同分辨率下，用户界面(如菜单、对话框、窗口和其他可视控件)布局、风格是否满足客户要求，文字是否正确，页面是否美观，文字、图片组合是否完美，操作是否友好等。

GUI测试的目标是确保用户界面会通过测试对象的功能来为用户提供相应的访问或浏览功能。确保用户界面符合公司或行业的标准，包括用户友好性、人性化、易操作性测试。

4. 用户体验易用性测试(UE Testing)

用户体验易用性测试主要是检测用户在理解和使用系统方面到底有多好，是否存在障碍或难以理解的部分。

用户体验易用性的测试方法，一般是通过用户访谈，或邀请内测、小范围公测等方式进行，通过不同实验组的运营结果来判断是否存在易用性缺陷。但由于缺乏有效的测试工具，必须需要大量的测试样本才能获得比较真实的测试数据，投入资源较多，测试周期较长。

参考GUI测试方法，为了更好地测试普通用户对App UE的反馈，可以进行用户易用性测试，找n组测试者(1组12人——军队一个班的建制，UE测试建议选取最大12组测试者、144人)，试用App的原型UE易用性，记录测试者的操作轨迹，当然包括严重的Bug。

(1) 邀请外部用户在现场对终端进行操作；

(2) 发布测试任务到指定社区，要求n组用户按照易用性测试要求，通过用户自己的终端进行相关连接操作，将完成的任务提交到指定社区。

通过此类测试，可以有效地发现不同用户操作App UE行为轨迹差异，以判断App的UE是否存在设计不恰当或许要改进的地方。

5. 安全性、访问控制测试(Security Testing)

移动设备/智能终端的安全性是一个需要考虑的重大问题，特别是在越来越多的业务功能和流程采用移动方式的情况下。IDC移动安全软件市场研究中特别强调了这一问题，研究发

现“50%以上的用户对其移动部署中的安全或合规性问题进行了报告”。

移动应用提供对信息的访问能力，并让用户能够像连接至物理网络一样完成敏感的事务处理。根据 AVG 在《电脑顾问》上的一篇文章称，“56%的智能手机用户的手机曾经丢失或被盗。”

想象一下，某家公司的高级管理人员在海外机场遗失了移动设备/智能终端。这个设备或终端是否会落入坏人手中？他们是否会访问设备或终端上的应用、网络和数据？如果在应用的要求和设计中未能解决安全性问题，则有可能遭受意外风险。

如果希望在设计中保证安全性，那么绝对需要一种测试和验证应用安全性的方法。

请充分考虑移动安全性的以下可能方面：应用如何管理身份验证通常是一个非常重要的考虑事项，旨在确保用户确实获得授权以访问信息。此外，存储在设备上的信息以及传输的信息必须受到妥善和充分的保护。如果有可能处理敏感性信息，那么必须采取适当级别的加密。请不要忽视未加密数据通过移动或 Wi-Fi 网络传输的风险。安全性应作为贯穿移动设备/智能终端应用开发和测试过程的主要考虑事项，以确保控制并减轻这些风险。

安全性和访问控制测试侧重于安全性的两个关键方面。

(1) 应用程序级别的安全性，包括对数据或业务功能的访问。应用程序级别的安全性可确保：在预期的安全性情况下，主角只能访问特定的功能或用例，或者只能访问有限的数据。例如，可能会允许所有人输入数据，创建新账户，但只有管理员才能删除这些数据或账户。如果具有数据级别的安全性，测试就可确保“用户 1”能够看到所有客户消息(包括财务数据)，而“用户 2”只能看见同一客户的统计数据。

(2) 系统级别的安全性，包括对系统的登录或远程访问。系统级别的安全性可确保只有具备系统访问权限的用户才能访问应用程序，而且只能通过相应的网关来访问。

由于 App 安全性、访问控制测试，通常需要设定不同的用户使用场景和多种变化事件，非常耗费人力、时间与资源。比较有效的执行方法，是将 App 安全性、访问控制测试任务进行分解，发布到指定社区进行群测，这样可以快速、成本可控地通过广泛的社会化专业测试人员的有偿参与，完成测试任务。

对于指定社区群测的测试者的测试过程，要求全程进行记录，监控、稽核测试结果的完整性与有效性。

6. 性能测试

我们还必须测试并解决移动应用出现的特定性能问题。最突出的一个问题是如何合理设计移动应用，使其通过性能不一的数据连接来运行。这不仅是设计时需要考虑的问题，也是在规划移动应用时需要考虑的关键因素。其原因在于，面对移动应用，用户对低迷的表现没什么耐心。据 Aberdeen Group 调查，“25%的用户在三秒延迟后就会放弃使用移动应用”。

应用和移动网站需要针对移动体验进行优化，同时还要适应移动设备/智能终端常见的带宽有限且多变的特性。移动应用另一个不甚明显的性能问题是共享的移动网络对应用性能的影响。

移动设备/智能终端会对现有系统产生惊人的影响。在现有系统中增加了移动访问后，移动设备/智能终端有可能让系统速度显著降低，甚至使系统崩溃。问题在于移动设备通常需要更长时间来完成事务处理，因而锁定了一般会快速使用并释放的关键服务器资源。

根据一项内部性能测试统计，只需几部运行缓慢的移动设备/智能终端就能让系统性能降低高达 200%～300%。对移动设备/智能终端应用进行性能测试是开发和测试流程的重要步骤，不容忽视。

管理他们的传统 IT 环境，但他们所采用的解决方案可能不足以应对移动性带来的全新复杂性。

为了继续提供相同或更高的服务质量，我们需要一个能够监控从应用到设备、运营商和后端基础设施的端到端移动业务服务运行状况的方法。所有这些元素都会对移动最终用户的体验产生影响，而我们肯定想在从 Twitter、Facebook 或网上其他地方读到用户相关反馈前，预先了解这些问题。

14. 修补和更新或开发与运维的协调

由于应用通常是在应用市场上部署，对我们的应用（以及品牌）的看法将极为透明，因此开发团队需要对在生产中发现的问题极为敏感。要测试所有可能的移动组合和情况不具可行性，坦白地说根本不可能，因此几乎可以肯定，补丁和更新必不可少。

如果开发和运维团队在各自的传统职责范围内孤立运作，那么发现问题和最终解决之间的时间间隔将超出大部分用户的忍受限度。在这种情况下，由于移动设备/智能终端应用更迭变化的速度很快，人们可以接受更为频繁地修补和更新应用。不论我们是否将这种模式称为“开发-运维”，对移动设备/智能终端应用而言，开发和运维团队都需要密切合作，共同监控已部署的应用并持续驱动未来的功能增强和漏洞修复，进而提升整体应用质量和最终用户体验。

15. 回归测试

回归测试是指修改了旧代码后，重新进行测试，以确认修改没有引入新的错误或导致其他代码产生错误的测试，回归测试为测试中重要的环节，是 App 发布、维护阶段，对缺陷进行修复后的测试。其目的是验证缺陷已经得到修复，检测是否引入新的缺陷。

15.2.3 如何开展移动 App 测试

测试人员的核心能力在于提出有挑战性的相关问题。这紧密地依赖前期的准备工作，如阅读、调查、询问、学习和运行等，并掌握所测产品的所有最新细节资料，这样才能全面地了解问题和有助于发现问题（问题可能来自对话、设计、文档、用户反馈或者是产品本身）。

1. 收集相关信息

这个阶段，测试人员可以问这些问题：

(1) 有哪些基础信息？如被测项目规格、项目会议、用户文档、知识渊博的团队成员、有无支持论坛或者是公司在线论坛提供帮助、有无现存 Bug 的记录。

(2) 该应用是在什么系统、平台和设备上进行运行和测试？

(3) 该应用是处理什么类型的数据（如个人信息、信用卡等）？

(4) 该应用有整合外部应用（如 API 和数据来源）吗？

(5) 该应用需要用到特定的移动端网页吗？

(6) 现有消费者如何评价这个产品？

(7) 有多少时间可用于测试？

(8) 测试的优先级和风险是什么？

(9) 哪些用户使用起来不愉快，为什么？

(10) 如何发布和更新？

(11) App 究竟是做什么用的？用户实际上是如何使用它的？

基于以上收集的信息，测试人员可以制定测试计划了。通常预算决定测试方法，一天测完、一个星期或一个月测完的方法肯定不同。当你逐渐熟悉团队、工作流程以及这类问题的解决方式时，你就更容易预测结果了。

2. 站在用户的角度找问题

测试人员可以不同的用户角色进行测试，这种把自己当成不同用户进行思考、分析和设想的能力对测试是很有启发的。如测试人员可能会设想自己是毫无经验、很有经验、爱好者、黑客、竞争对手等用户角色，当然还有更多可选的角色，这主要取决于被测产品是什么。

除了角色特点外，其操作行为和工作流程也很重要。人们使用产品方式常常很奇怪，如：在不应该返回的时候返回了，不耐心而且多次敲按键，输入错误的数据，不理解该么做，可能没有按要求进行设置，可能会自以为是地认为自己知道该做什么(比如通常不阅读说明)等。测试人员遇到这些问题时，也常常发现意料之外的 Bug。有时候，这些 Bug 微不足道，但是更深入的调查就会发现更严重的问题。

很多问题是可以被预先确定和测试的。测试移动端 App 时，有些问题并不都有关，但是也可以尝试问问，如：

(1) 是否按照所说的来做呢？

(2) 是按设计完成任务的吗？(或不是按设计完成任务的吗？)

(3) 如果处于一直被使用或者负荷情况下(或运行到极限时)，状况会怎么样？会反应迟钝吗？会崩溃吗？崩溃报告会反馈到 App 吗？会更新吗？有反馈吗？

(4) 用户可能有哪些创造性的、逻辑性的或是消极的导航方式？用户相信你的品牌吗？

(5) 用户的数据安全如何？

(6) 有可能被中断或是被破解吗？

(7) 会要求打开相关服务(如 GPS、Wi-Fi)吗？如果用户打开会怎样？没打开又会怎样？

(8) 将用户重新引向哪儿？去网页，还是从网页到 App？这会导致问题出现吗？

(9) 沟通过程和市场反馈是否符合该 App 的功能、设计和内容？

(10) 登录流程是怎样的？能在 App 上直接登录还是要去网页端登录？登录是否整合了其他服务，比如用 Facebook 和 Twitter 账号登录？

3. 从数据中找问题

测试人员要形成从数据中找问题的好习惯。事实上，用户或者是软件开发人员在信息流中确实太容易迷惑了，因为可能出现很多错误，所以基于数据和云的服务更为重要。我们可以尝试在以下场景中检查出问题：

(1) 移动设备/智能终端数据已满；

(2) 测试人员移除了所有的数据；

(3) 测试人员删除了 App，那数据怎么办？

(4) 测试人员删除并重装了 App，数据怎么办？

(5) 过多或者过少的内容导致设计和布局的改变；

(6) 在不同的时间段和时区使用；

(7) 数据不同步；

(8) 同步被中断；

(9) 数据更新影响其他的服务(比如网页和云端服务)；

(10) 快速处理数据或是处理大量的数据；

(11) 使用无效的数据。

测试人员也很喜欢测试极限数据下的情况。他们常常是作为典型用户来了解这个 App，所以极限下的测试并不会花很长的时间。数据是混乱的，所以测试人员要考虑软件的用户类型，以及在不同的数据场景下如何进行测试。

比如，他们可能尝试以下场景：

(1) 测试用户可输入的极限值；
(2) 用重复的数据进行测试；
(3) 在全新无数据的手机里测试；
(4) 在老手机上测试；
(5) 预先安装不同类型的数据；
(6) 考虑聚集大家的资源来进行测试；
(7) 让一些测试自动化；
(8) 用一些超出预期的数据去测试，看它是怎么处理的；
(9) 分析信息和数据是怎么影响用户体验的；
(10) 不管用户看到的是否正确，都要一直问问题。

4. 从出错信息中找问题

这里，我们不是从设计师的角度来谈论好的错误消息的设计，而是想从用户或是测试者的角度来看这个问题。出错信息是测试人员很容易发现问题的地方。

关于错误信息可考虑以下问题：

(1) 出错提醒的用户界面 UI 设计可以接受吗？
(2) 错误信息内容可以理解吗？
(3) 错误信息是否保持一致？
(4) 这些错误信息有帮助吗？
(5) 错误信息内容是否合适？
(6) 这些错误是否符合惯例和标准？
(7) 这些错误信息本身是否安全？
(8) 运行记录和崩溃是否能被用户和开发者获得？
(9) 是否所有的错误都被测试过？
(10) 用户处理完错误信息后，将处于什么状态？
(11) 是否在用户应该接受错误信息时，却没有错误信息弹出？

错误信息会影响用户体验。然而，不好或无用的出错提醒无处不在。最理想的状态是避免用户遭遇错误信息，但这几乎不可能。出错情况的设计、实现和确认可能与预期相反，但是，测试者往往善于发现意料外的 Bug，并能仔细考究是否改进它们。

5. 从平台中找问题

对于任何项目团队成员来说，了解相关平台的业务、技术和设计上的限制，都是至关重要的。移动端 App 的测试人员应该找出这些平台相关的问题：

(1) 是否遵照了这个特定平台的设计规范？
(2) 与竞争对手以及行业内的设计相比如何？
(3) 是否适应外围设备？
(4) 触摸屏支持手势吗，如轻拍、双击、长按、拖动、摇动、夹捏、轻拂、滑动？
(5) 这个 App 可以被理解吗？
(6) 当转动设备的方向时，有什么变化？
(7) 可以使用地图和 GPS(全球定位系统)吗？
(8) 有用户指南吗？
(9) 电子邮件的工作流程友好吗？

(10) 通过网络分享时,它运行得流畅吗? 是否整合了其他社交应用或网站?
(11) 当用户正在进行多任务工作,并在不同 App 间切换的时候,它还运行正常吗?
(12) 当用户更新它时,它是否会显示时间进度?
(13) 默认设置如何? 有经过调整吗?
(14) 使用音效会有不同吗?

6. 从连接和中断中找问题

连接和中断会产生很多问题,如当连接断断续续或是意外中断时,很多问题就可能发生。

可尝试着这样使用 App:走动环境下,Wi-Fi 连接下,没有 Wi-Fi 的情况下,3G 模式下,间歇性地连接,设置为飞行模式,一个电话打进来时,接收到一条信息时,接收到一个提醒通知时,在电量很低甚至自动关机时,被强制更新时,收到一条语音留言时。

这类测试最容易发现错误和 Bug。因此,需要在以下情况下进行测试(不仅仅只是开机、确认它可以正常工作,还要尝试用户使用的整个流程,并在特定的时间间歇内强制连接和中断):

(1) 这个 App 提供了足够多的反馈吗?
(2) 数据传输为用户所知吗?
(3) 它会慢慢停止,然后崩溃吗?
(4) 开启时会发生什么?
(5) 任务完成中会发生什么?
(6) 是否可能丢失未保存的操作?
(7) 你可以忽视通知提醒吗? 忽视后会发生什么?
(8) 你可以对通知提醒做出响应吗? 响应后会发生什么?
(9) 对某些问题,使用错误信息是否恰当?
(10) 当登录过期或超时会发生什么?

7. 从环境变化中找问题

当外界环境持续变化时,App 又会受到哪些影响呢? 如:

(1) 我可以下载这个 App 吗?
(2) 我可以下载并安装更新吗?
(3) 更新之后还能使用吗?
(4) 当很多 App 处于等待更新状态时,我能更新它吗?
(5) 系统更新后,它会发生什么?
(6) 系统未更新,它又会发生什么?
(7) 它会通过 iTunes 自动同步下载到其他设备吗?
(8) 它自动执行任务或测试有意义吗?
(9) 它会连接到网络服务吗? 这会带来什么不同?

移动端的 App 每一个版本发布后,最好都去测试一下。每次发布新版本时,先定义最高优先级测试,确保其能在各种条件下进行(主要是在主流的平台上)。随着时间的推移,测试可以变得自动化。

习题

1. 简述移动应用的特点和范围,对于移动应用通常用户都有什么样的要求?
2. 移动应用测试的问题或困难都有哪些? 通常是怎样开展移动应用的测试的?

第16章 云应用软件测试

云计算概念提了很多，大家也越来越关注云应用，云计算真正的发展也是云应用。云应用对于企业来说无疑是利好，云能快速响应企业变化的需求，即需即用。目前一些地方和企业已经有相关的云应用了。当然云应用也会面临一些挑战，如安全性与隐私性保护的问题。另外，阻碍云计算广泛使用的一个重要因素就是如何确保云计算服务本身的质量以及如何更好地调用云计算的资源。因为前者是后者的保障和基础。对于前者而言，本身的软件质量关系到云计算服务的可靠性。然而，云计算是一个复杂、动态、分布式的体系结构，并且存在大量异构软件系统的协同工作，极易隐藏一些潜在的错误，多个系统之间存在频繁交互或大规模数据流动，这些都需要测试加以保证；后者则是云计算资源的调度优化，关系到云计算特性的充分发挥，这些均与软件测试密切相关。由此，云计算的出现必然会给传统软件的测试方式带来深刻变革，这就意味着测试从桌面软件转向在线软件服务，而相关软件工程测试的方法、工具以及概念都会因此发生变化。因此，要让企业和组织大规模应用云计算技术与平台，就必须分析并着手解决云计算所面临的各种问题，测试就是一种非常有效的保证手段。软件测试在云计算的背景下，面临着前所未有的挑战。

当然中国巨大的市场也在促进着云应用的实现，未来将出现一批成功的云应用案例。

16.1 云测试基本概念

云计算时代的到来为软件服务提供广阔的平台，软件测试就是其中之一。我们可以借助于云计算平台的基础设施建立测试环境，借助于云计算平台的软件测试工具进行软件测试，当然也可以借助于该平台获得软件测试服务提供商提供的专业的测试服务，这种基于云计算的新型的测试方式就是云测试。云测试是随着虚拟机技术、云计算技术的发展应运而生的测试方案。它可以以按需、易扩展的方式向用户交付所需的资源，包括基础设施、应用平台、软件功能等服务。

“云测试”由测试和云两者组成。首先，它是一种软件测试，有软件测试常规或传统的测试手段、测试方法、测试过程；其次，它应该工作于“云端”，通过云来实现其方法、过程。由以上两点可知，“云测试”就是通过“云”而实施的一种软件测试，由于与云的结合，所以它在测试方法、手段、过程等方面，具有一些自己独有的特征。即云测试是基于云计算的一种新型测试方案。服务商提供各种硬件设备、软件系统、测试工具、服务等，用户只需制定好测试方案、编写好脚本，就可以在云测试平台上完成软件的测试。

16.1.1 云测试特点

由于“云测试”基于云计算之上，所以“云测试”的特征亦应与“云计算”密切相关。所以先

来看看云计算的特征。云计算作为一种新的计算方式，它区别于传统计算的特点，就是它的计算资源分布在云端，用户通过访问云来使用计算资源，而不用关心计算资源的数量、结构、运行状态、成本、升级、运维等。

1. "云计算"的特征

"云计算"具有以下特征。

(1) 计算资源的服务化。所有计算资源以一种服务的方式提供，用户通过购买服务，来使用计算资源，所提供服务具有统一的接口、统一的表示方式，通过服务的方式，将计算资源封装起来，其技术、架构、运维、建设、升级等都对用户不可见，用户也无须关心，这也是"计算资源将来如同自来水一样，打开水龙头就可使用"这一白话式描述的由来。

(2) 计算资源的虚拟化。为了让计算资源能够让用户随需使用，根据其使用需求来动态扩展，最好的办法是将计算资源虚拟化，通过虚拟化使物理的计算资源柔性化，可以动态延展，随需改变，如同天空的云一样，万般变化。

根据"云计算"的特征，我们可以看到"云测试"的特征。

2. "云测试"的特征

"云测试"具有以下特征。

(1) "测试资源"的服务化。软件测试本身以统一接口、统一表示方式实现为一种服务，用户通过访问这些服务，实现软件测试，而不用关注"测试"所使用的技术、运行过程、实现方式等。比如，你要对你的某个软件进行测试，你只需提交你的软件，提交的方式可能是源代码、可执行文件，或者已经部署好的系统，然后就可以访问云测试服务，直接执行测试，并获得测试结果。

(2) "测试资源"的虚拟化。云计算的虚拟化实现方式，为云测试的虚拟化提供了较大的便利，测试资源的虚拟化，使测试资源可以随用户的需求提供，动态延展。

16.1.2　云测试优点

在云平台上进行的测试，与传统的测试类似，包括功能测试、性能测试、安全性测试；能够进行自动化的功能测试与回归验证；测试过程包括测试用例的设计、测试问题的提交、测试计划、测试报告以及测试管理等工作。但是相对于传统的本地测试来说，云计算平台的测试具有本地测试不具有的优越性。云测试主要具有以下几个优点。

(1) 不需购买昂贵的测试工具，只需支付低廉的租赁费用，降低企业成本。通过服务实现测试，而不用自行购置测试工具、测试环境等，成本会大大降低。

对于企业来说，应用云测试不需要购买或准备多台测试服务器和个人电脑，购买各类价格昂贵的测试软件；也不再需要部署复杂的测试环境。只需要列出测试的目的、测试环境要求、虚拟机台数、何时间断租用即可，实现按需付费。同时随着企业软件版本更新和软件开发技术的更新，被测试的软件或测试环境也需要不断地升级换代，会进一步产生升级和维护费用。而在云测试环境中这些因素企业都无须考虑，交由提供云测试服务的供应商完成即可。

(2) 云测试服务商给企业提供超大规模的测试资源、动态分配、在线支持、提高测试效率。对于一些大规模系统，数据量、用户量庞大，如用户量庞大的游戏系统，采用传统测试手段，需要构建一个庞大而复杂的测试环境，成本高昂，难以实施。而云测试中的测试资源虚拟化就为其提供了便利，根据测试需求动态扩展测试资源，大大便于大规模系统测试的实施。

云测试的运用减少了测试环境搭建时间，例如测试机器及网络的准备、操作系统安装、各种测试工具软件安装等。测试人员只需提前将需要配置的测试环境通知云测试服务商，到时

间直接使用即可。同时,测试过程中遇到软件使用等问题,也可以获得云测试服务商的远程在线支持,实时响应速度很快,不会出现测试暂停的状况。

(3) 云测试能够为测试人员提供各种系统平台环境。云测试环境中,能为测试人员模拟不同系统平台的运行环境,而这些环境不需要测试人员自己搭建,通过云测试提供者提供的服务就能方便地实现被测系统在不同系统平台的执行结果。

(4) 在云测试中,测试资源一旦得到申请立即可以使用,不需要进行复杂的准备过程。云测试提供了一整套的测试环境,测试人员利用虚拟桌面等手段就可以登录到云测试环境进行测试。这使得测试人员不必自己进行软硬件的安装、测试环境的配置以及测试环境的维护,这些都能由云测试的提供者来提供相应的服务。就目前的虚拟化技术来说,测试人员可能不需要创建自己的测试环境,通过利用标准化的云测试环境来进行测试。或者只需要几小时就能创建一套新的测试环境,大大减少了测试的准备时间。

当测试用例非常庞大或是反复运行测试时,工作量就会显得非常巨大。利用"云"可以并发地运行测试用例,加速测试进程。

(5) 云测试能够为测试人员提供各种便捷的服务。云测试除了可以给测试人员提供完整的测试环境,还可以提供许多测试服务。例如,云测试可以提供还原点将虚拟机重置到指定状态;在测试执行过程中,能监控各种资源,帮助测试人员发现问题,定位错误;云测试还可以提供多台测试客户机,进行大规模的模拟测试,通过云测试平台的各种工具为测试人员提供便捷的服务。此外,云测试平台还可能提供专业的专家服务知识,使测试人员能获得专家级的详细测试分析结果。

(6) 测试知识的复用。云测试中会积累大量的测试知识资源,每个用户可以通过云端系统将自己的测试知识共享给其他用户,并通过其他用户的使用获利。

(7) 其他优势:①真实模拟。云测试能够更加真实地模拟分布式的虚拟用户环境,包括地理位置、浏览器、操作系统、网络宽带等特性,避免模拟用户形式的单一。②按需提供。测试过程中,不了解测试需求,导致测试资源的极大浪费。而云测试提供了一种按需测试的方式,用户可以灵活按需地部署测试资源、环境,当测试完后可以释放相关资源。③早期测试。可以尽早测试,不必最后才进行单元、集成、系统、负载、压力等测试,完全可以在开发的前期进行,提高了效率。④加速测试。当测试用例非常庞大或是反复运行测试时,工作量就会显得非常巨大。利用"云"可以并发地运行测试用例,加速测试进程。⑤规范标准。云测试提供了一种不同应用系统测试的共同解决方案,可以预先定义多种测试所需环境的标准镜像文件。在某种意义上来说,这必然促进不同应用系统间标准的统一化,从而加速了云技术的发展。

目前,网络上许多的"云测试",都是各个厂商利用原有的工具、产品,稍加变化,以 B/S 结构的方式展现出来,但在测试资源服务化、测试资源虚拟化等方面,还没有实现,仅仅是披了一个"云"的外衣。但无论怎样,这样总比没有要好。

16.2 云测试方法和技术

从前面"云测试"特点所涉及的内容来看,可以认为云测试是一种有效利用云计算环境资源对于其他软件进行的测试,或是一种针对部署在"云"中的软件进行的测试。

16.2.1 云环境中的测试和针对“云”的测试

云测试的过程中经常会同时涉及在云环境中的测试和针对“云”的测试，比如部署在云环境中的软件需要进行测试，而此测试又要调用云计算环境的资源，这就同时涉及上面提到的两个方面。

1. 在云环境中的测试

在云环境中的测试主要是利用云资源对其他的软件系统进行测试，涉及与云测试密切相关的资源调度、优化、建模等方面问题。以便为其他软件搭建廉价、便捷、高效的测试环境，加速整个软件测试的进程。在这一类型的测试中，其他的软件可以是传统意义上的本地软件，也可以是“云”中的应用软件服务。而且，云计算作为一种可以快速获得的有效资源，已经参与到软件测试的各阶段中，云计算能够快速配置所需测试环境，此种转变必然会给传统测试方式带来变革。

2. 针对“云”的测试

针对“云”的测试涉及云计算内部结构、功能扩展和资源配置等多方面测试问题。测试部署在云环境中的各种云计算软件。在针对“云”的测试中，各层的云服务对一般服务用户是透明的，它由大量动态、异构、复杂的系统构建，并且随着业务需求的变化，系统还在不断更新和演化，这必然导致很多隐藏的错误不容易被发现，因此一般需要考虑如下几个主要方面的测试内容，如图16-1所示。

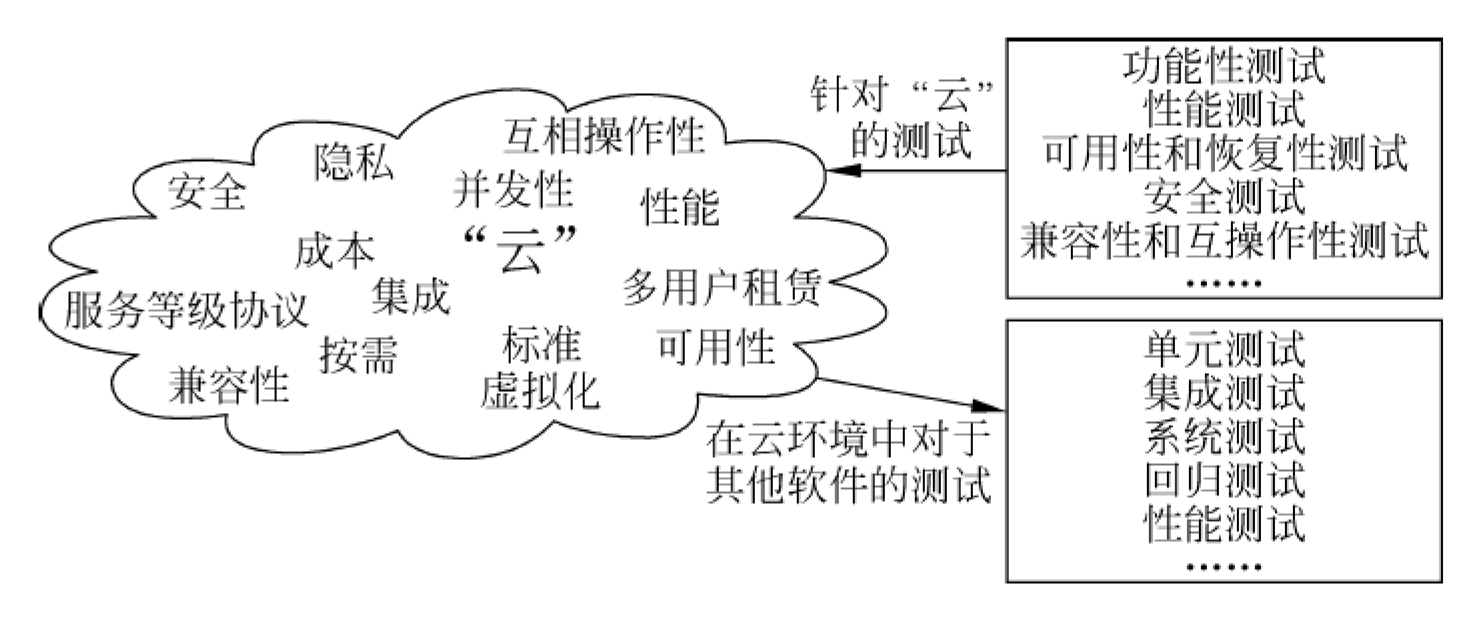


图16-1 云测试内容

1）功能性测试

功能性测试与传统的软件类似，主要包括单元、集成、系统测试等内容，是确保开发的云服务功能能够满足用户需求。

2）性能测试

性能测试包括压力和负载测试，测试云服务的性能能否满足用户按需服务的要求。

3）可用性和恢复性测试

可用性和恢复性测试主要针对发生灾难性事件后，“云”中的数据能够在较短暂的时间内快速恢复，使得云服务的可用性较高。

4）安全测试

安全测试是为了确保云服务中存储、流动数据的保密性、完整性。“云”的安全是云服务能否推广使用的关键。

5）兼容性和互操作性测试

兼容性和互操作性测试是为了确保开发的云服务能够运行在不同的配置环境下，如不同

的操作系统、浏览器、服务器等。

3. 迁移测试到“云”中

迁移测试到“云”中是指迁移传统的测试方法、过程、管理、框架到云环境中。迁移测试到“云”中包含在图 16-1 所示的两类测试中，这既有第一种云环境中的测试，也含有第二种针对“云”的测试问题，是两者的交叉。前者是指利用云环境测试其他软件，解决以往传统测试中资源获取的局限性问题，后者是指迁移传统的测试方法到“云”中，解决部署在云计算中软件的测试问题。

4. 云测试场景

由于云计算的特点，云测试一般会牵涉到测试者、本地应用系统、多个云、本地测试服务器等多个方面，其核心测试场景如图 16-2 所示，测试者可以通过调用不同的模块进行测试。其中编号①③④代表不同类型的云，而②⑤则分别表示传统意义上本地的应用系统和测试服务器。

(1) 从图 16-2 可以看出，传统的软件测试只局限于②⑤两个部分；

(2) 当本地测试服务器②不能满足进一步的测试需求时，通过本地测试服务器②和“云”④的混合使用，测试本地应用系统⑤，甚至测试可以不用本地测试服务器②，完全借助“云”④快速构建测试本地应用系统⑤的环境，这些测试均属于在云环境中的测试；

(3) 当针对“云”进行测试时，通过专门的云测试服务④分别测试“云”①或“云”③中的应用软件，由于“云”①③中的软件处在云环境中，并且使用的是测试服务④，所以对于它们的测试自然也是云环境中的测试，同时也是针对“云”的测试，而本地测试服务器②也可参与到对云应用的测试中；

(4) 传统的针对本地应用系统的测试方法，如单元测试、集成测试、回归测试、性能测试均可以考虑迁移到“云”中，云计算的特性必将给这些测试方法带来新的发展，如模块④⑤所示。

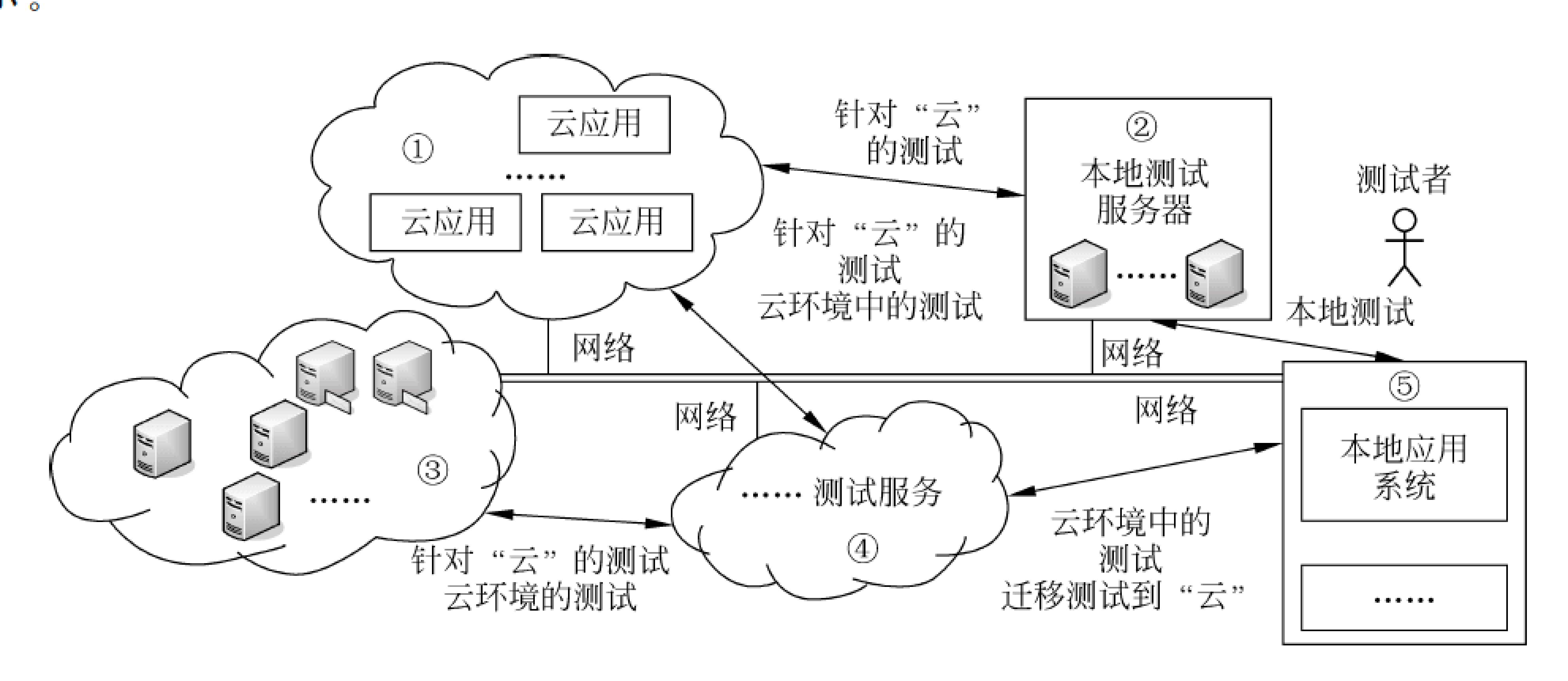


图 16-2 云测试场景示意

5. 云测试的执行过程

云测试执行初期我们要考虑以下四方面的问题。

(1) 了解云计算系统的测试需求及测试技术。对于云测试系统来说，测试需求关键是找出云计算系统有哪些风险。首先要求测试工程师对云计算有一个较深的理解，并对云计算的关键技术有一定了解，例如虚拟化技术、分布式的编程模式、存储模式、海量数据的管理、云平台自身的管理以及资源调度等，才能准确判断云计算系统测试的缺陷、漏洞和风险。

(2) 制定一份详细的云测试计划。与内部测试相比,云测试计划应更加详尽。因为一旦测试提交到云中,测试过程很难直接掌控,测试的风险也会加大。因此,详细的测试计划能减少这一过程带来的风险,一个标准的云测试计划的目标就是在测试执行的过程以及性能和安全性等各个方面降低风险。

(3) 确定云测试过程的安全性。在签署云测试协议之前,对云测试服务提供商的安全性做法和策略应有相当的了解,在云测试执行过程中也要确保对测试执行过程及结果分析的安全性,保证数据不被泄露。

(4) 云测试的执行。在确定好以上几点后,就可以在云测试平台上执行云测试了。

16.2.2 云测试抽象模型

目前云测试方案,各个厂家都是围绕自己的产品展开,我们将它抽象成一个云测试抽象模型,如图 16-3 所示。

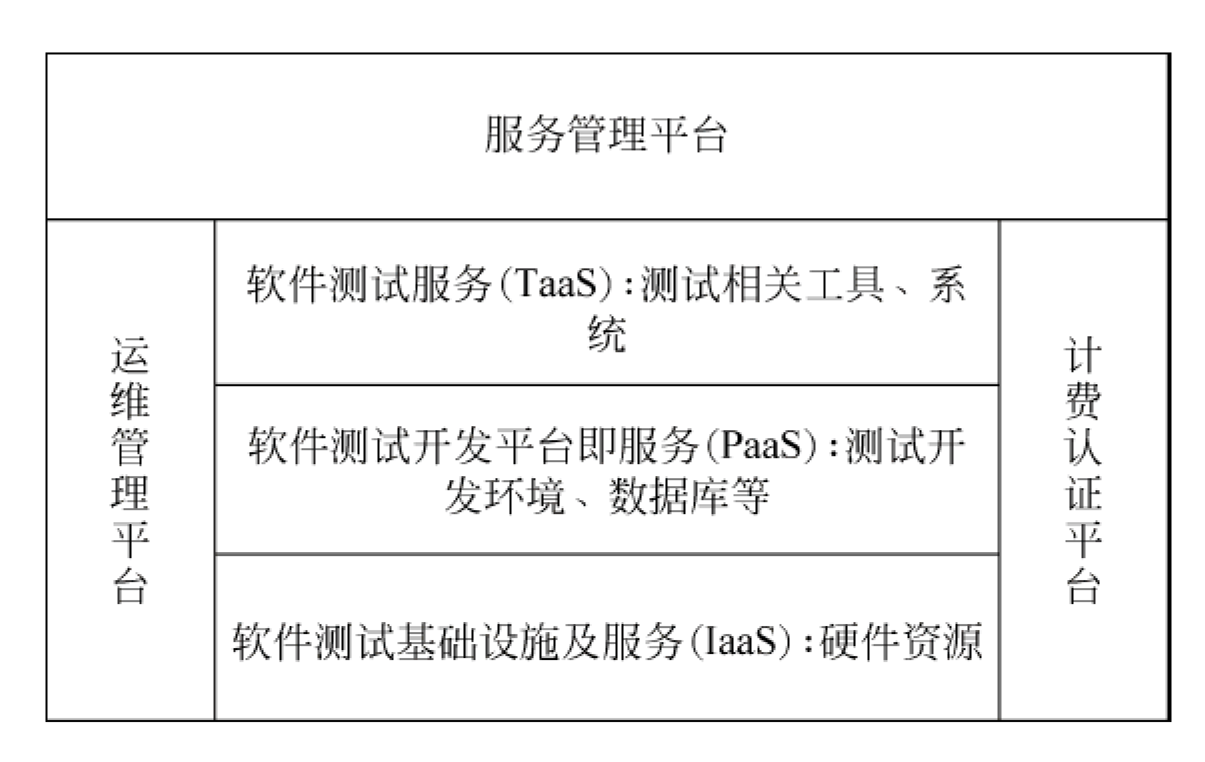


图 16-3 云测试抽象模型示意

该模型包括 6 个部分。

(1) IaaS:是把计算、存储、网络以及搭建测试环境所需的一些工具当成服务提供给用户,使用户能够按需获取 IT 基础设施;

(2) PaaS:把测试软件的开发、测试和部署环境当做服务,基于互联网提供给用户,为用户提供数据库、操作系统、测试开发环境等;

(3) TaaS:是一种基于互联网提供软件测试服务的应用模式,用户可在线使用各种测试服务,如测试自动设计、自动化功能测试、测试管理等;

(4) 计费认证平台:实现用户认证,提供各种服务层面的费用计算,支持多种计费方式;

(5) 运维管理平台:支持所有资源以及活动的自动监控和管理,使得几个人能轻松地管理数千台的物理设备;

(6) 服务管理平台:运营云测试服务的平台,可实现从服务请求、监控管理到服务结束的所有活动,是一个自动化的管理系统,可以管理云测试的所有资源及服务。

16.2.3 云测试现状及挑战

1. 云测试现状

目前云测试主要应用于以下三个方面。

(1) 测试人员利用云测试服务商提供的测试环境,运行自己的测试用例。

(2) 云测试服务商为测试人员提供测试执行的服务。测试人员编写好测试用例后,提交

给云测试平台，云测试平台执行测试并返回测试结果。例如常见的性能测试，测试人员需要将测试用例、虚拟用户数、网络连接配置等性能参数提供给云测试平台，云测试平台通过性能测试软件，例如 LoadRunner 来执行测试，并生成性能测试报告。

（3）测试中需要使用软件工具或测试运行于不同测试环境都可进行云测试。例如：测试软件在不同硬件环境平台下的运行；测试软件运行于不同操作系统、数据库环境，浏览器对平台的适应性；测试软件运行在安装不同防火墙及防病毒软件环境时运行的可靠性；自动化的功能测试以及性能测试等都适用于云测试随着云计算技术的发展。云测试提供商提供的服务越来越多，适合于云测试的项目也将不断增加。

2. 云测试挑战

云计算具有众多的优势，不可避免地对测试带来了极大的冲击与挑战，体现在以下几个方面。

1）数据安全

用户数据都是基于云环境的，会涉及用户敏感数据的隐私问题；同时随着应用信息的交互，这些数据会在不同系统之间流动。所有这一切都需要通过测试来保障数据的安全性。

2）集成问题

云计算软件系统必然是由多个异构系统构成的，提供用户不同的云计算服务，满足了用户需求，但也增加了系统的复杂性。而这些异构系统彼此间很难获得对方的代码，加大了集成测试的难度。

3）多用户租赁

云平台上的云应用是多用户租赁环境下的应用系统。多个用户共享一个实例化的应用实体及数据达到个性需求的目的，这就要求用户能够正确完成自身的操作功能，而彼此间的并发操作不会产生相互影响，对测试而言是一种极大的挑战。

4）服务保障

尽管云服务推崇的是资源和性能的可扩展性、可用性，但实际中，比较著名的云厂商（如 Amazon、Google 等）也出现过由于故障（如响应时间延长、网络带宽等）导致服务不可用的情况，这大大降低了人们使用云服务的热情。如何构建这样的一个可用性测试环境显得比以往更为复杂。

5）并发问题

云服务可以迅捷地提供测试其他软件所需的资源和环境，但并不是所有的测试过程和场景都适合云测试框架，需要考虑系统间、测试用例间相互的依赖关系。

6）兼容和交互性

云计算中的软件运行在多个不同环境中，那么测试比以往都要复杂，测试的环境显得更加不可控制，需要考虑"云"中软件和不同环境的兼容以及与其他"云"的兼容问题。

7）虚拟化问题

虚拟化技术提高了资源的利用效率，然而并不是所有的测试方案都支持虚拟化技术；同时，在一台机器上产生的多个虚拟设备存在资源的竞争机制，这样测试的结果可能会与实际有一定的偏差。

16.2.4 云测试平台

云测试平台提供一整套测试环境（测试基础设施、测试工具应用等），测试人员登录到该测试环境，就可以立即展开测试。这将软硬件安装、环境配置、环境维护的代价转移给云测试提

供者，极大地减少了测试环境搭建时间，如机器和网络准备、操作系统及各种测试工具软件安装等，提高了测试效率；在云测试平台上进行性能测试，可以开启更多的客户端，获得更加强大的运算能力，能够尽早发现和应对意料之外的流量高峰，让测试软件获得巨大的性能改善。图 16-4 所示为云测试平台功能结构图；图 16-5 所示为云测试平台架构图；图 16-6 所示为云测试平台各管理流程图；图 16-7 所示为云测试平台应用流程图。

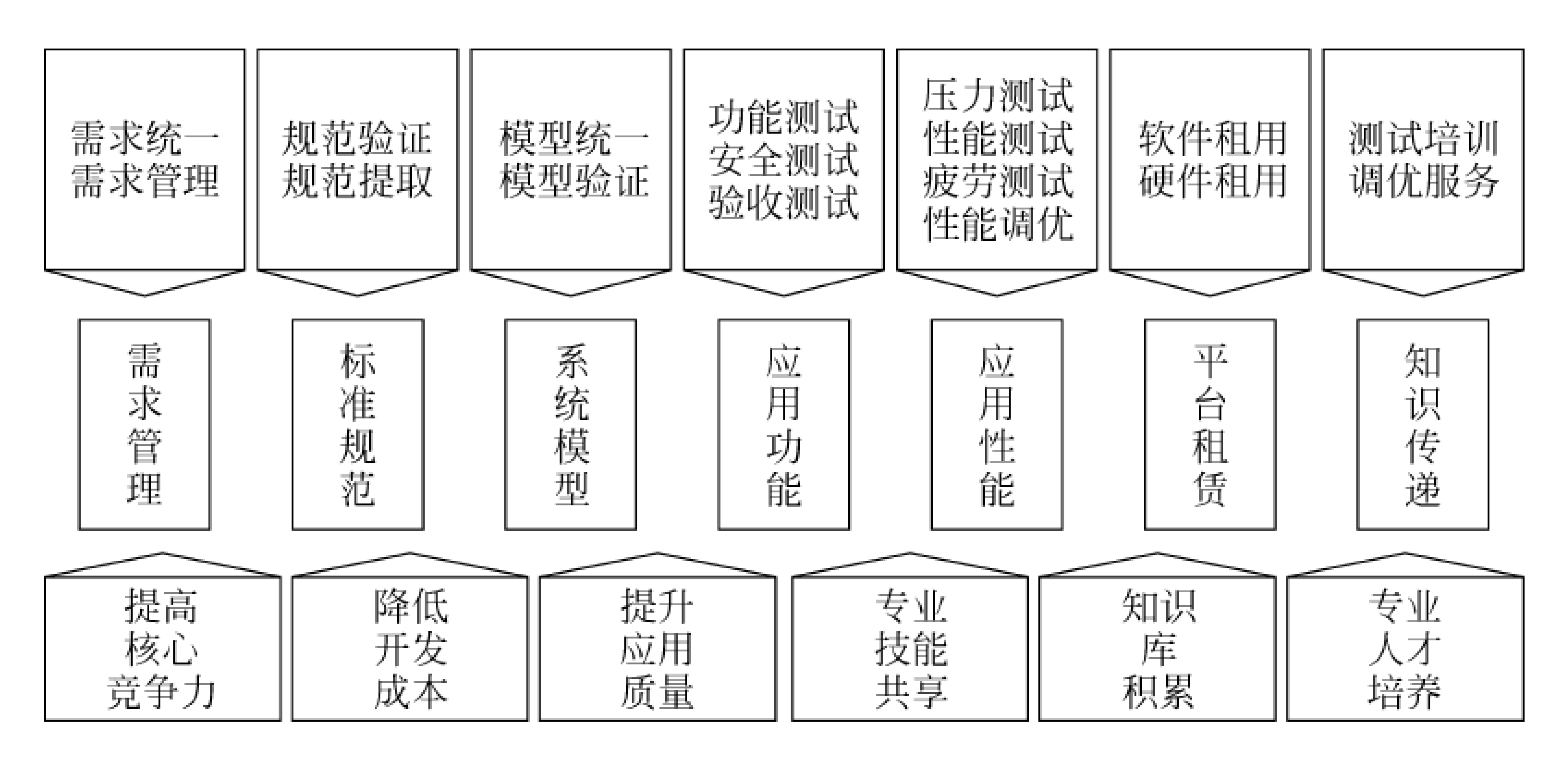


图 16-4　云测试平台功能结构图

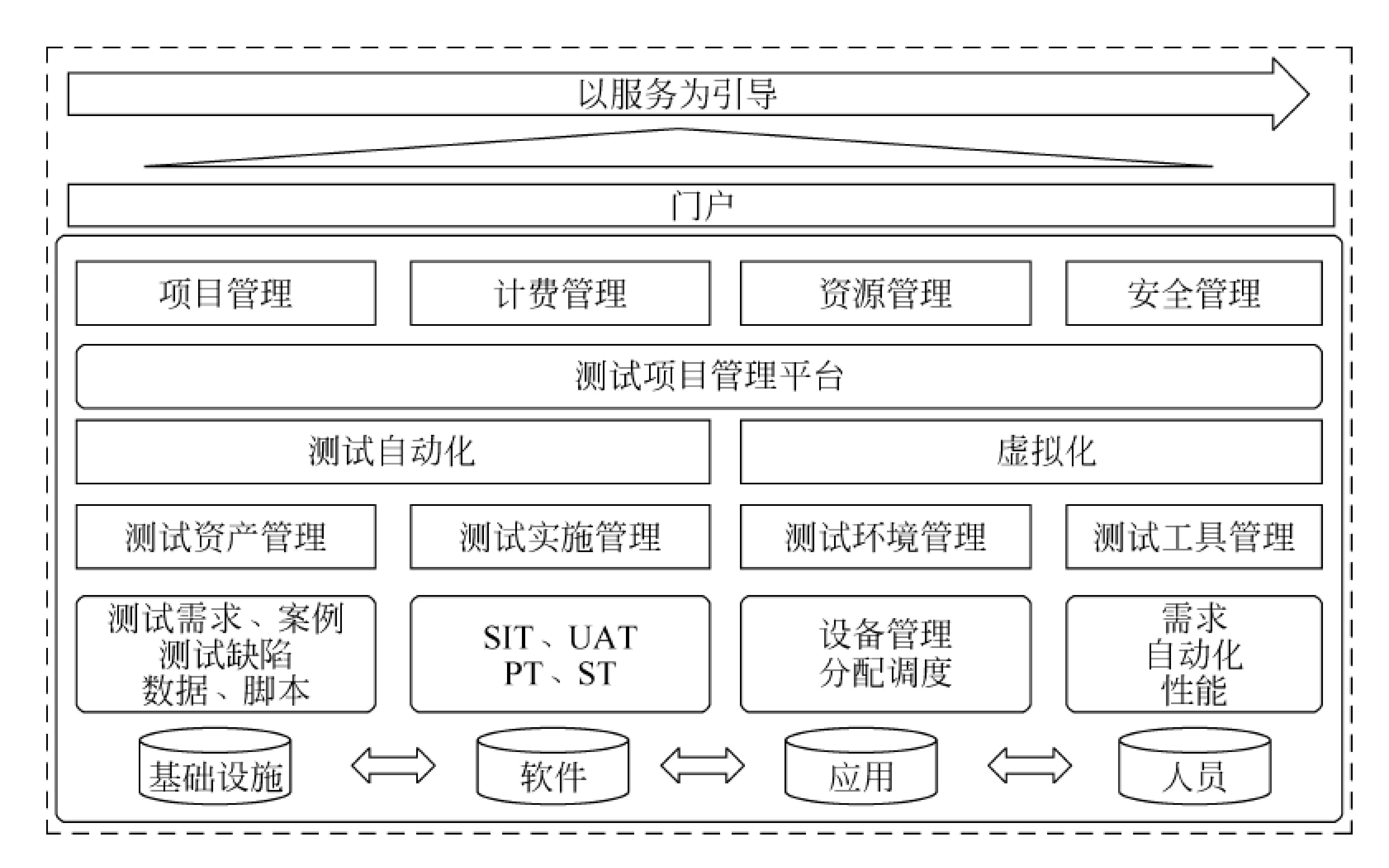


图 16-5　云测试平台架构图

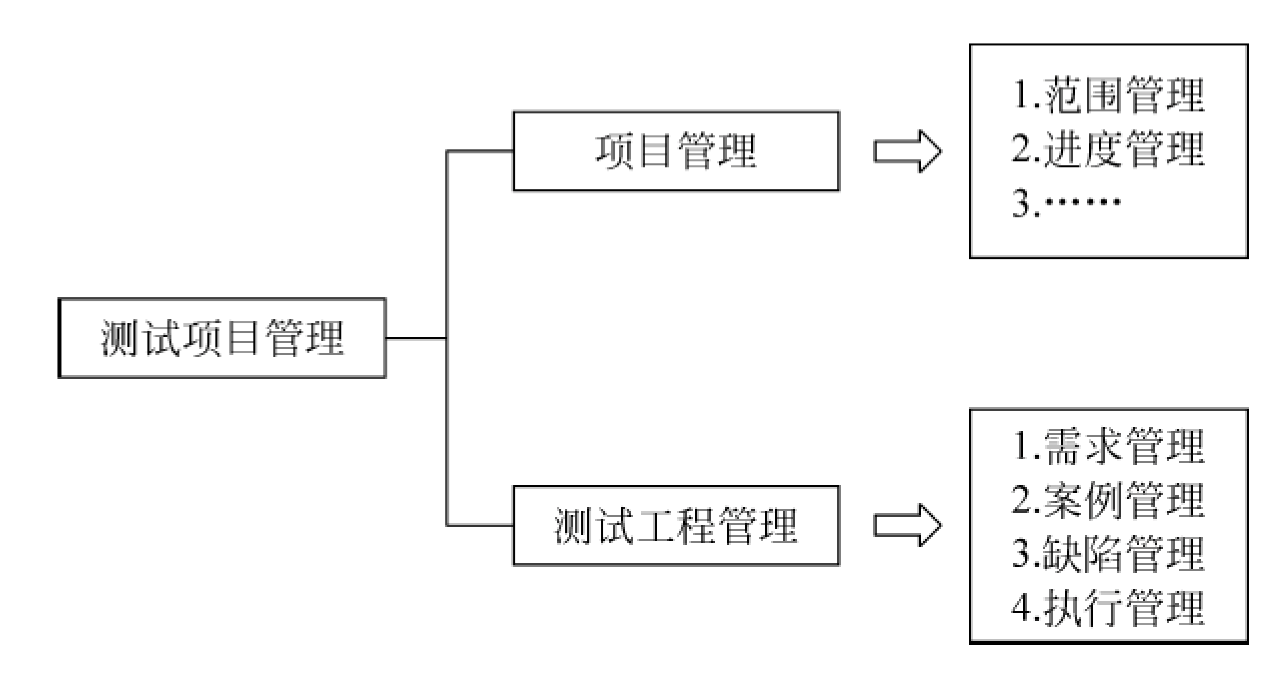


图 16-6　云测试平台各管理流程图

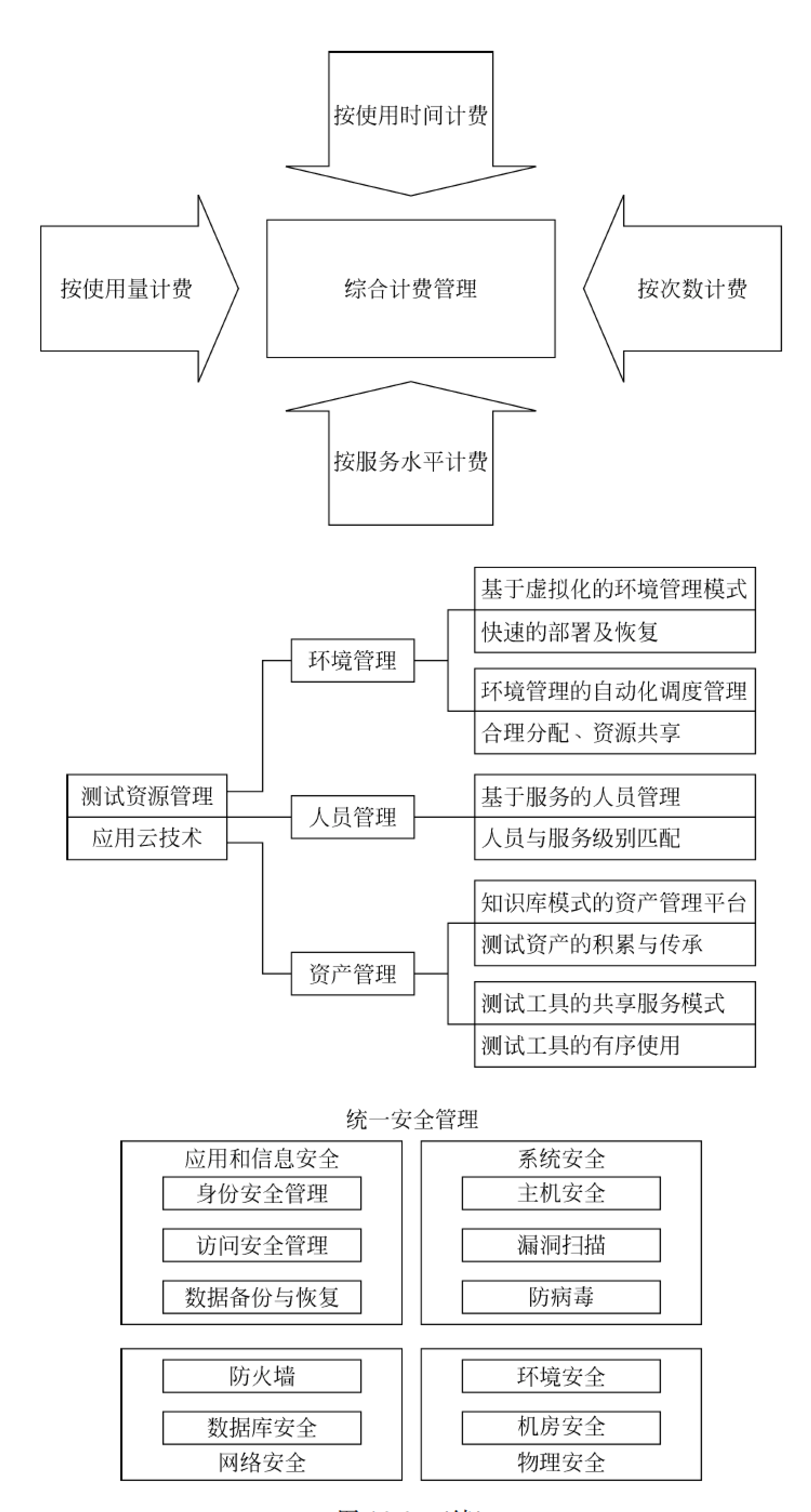


图 16-6 （续）

1. 云测试平台的特点

云测试平台的特点有：①新型的测试增值服务(云测试平台提供测试设备的设置、部署与维护服务，而对云的用户收取服务费的模式)；②广泛的测试服务类型(包括功能测试、性能测试、安全测试、验收测试、性能调优等)；③云平台消费模式(云用户只需前期支付一次性的项

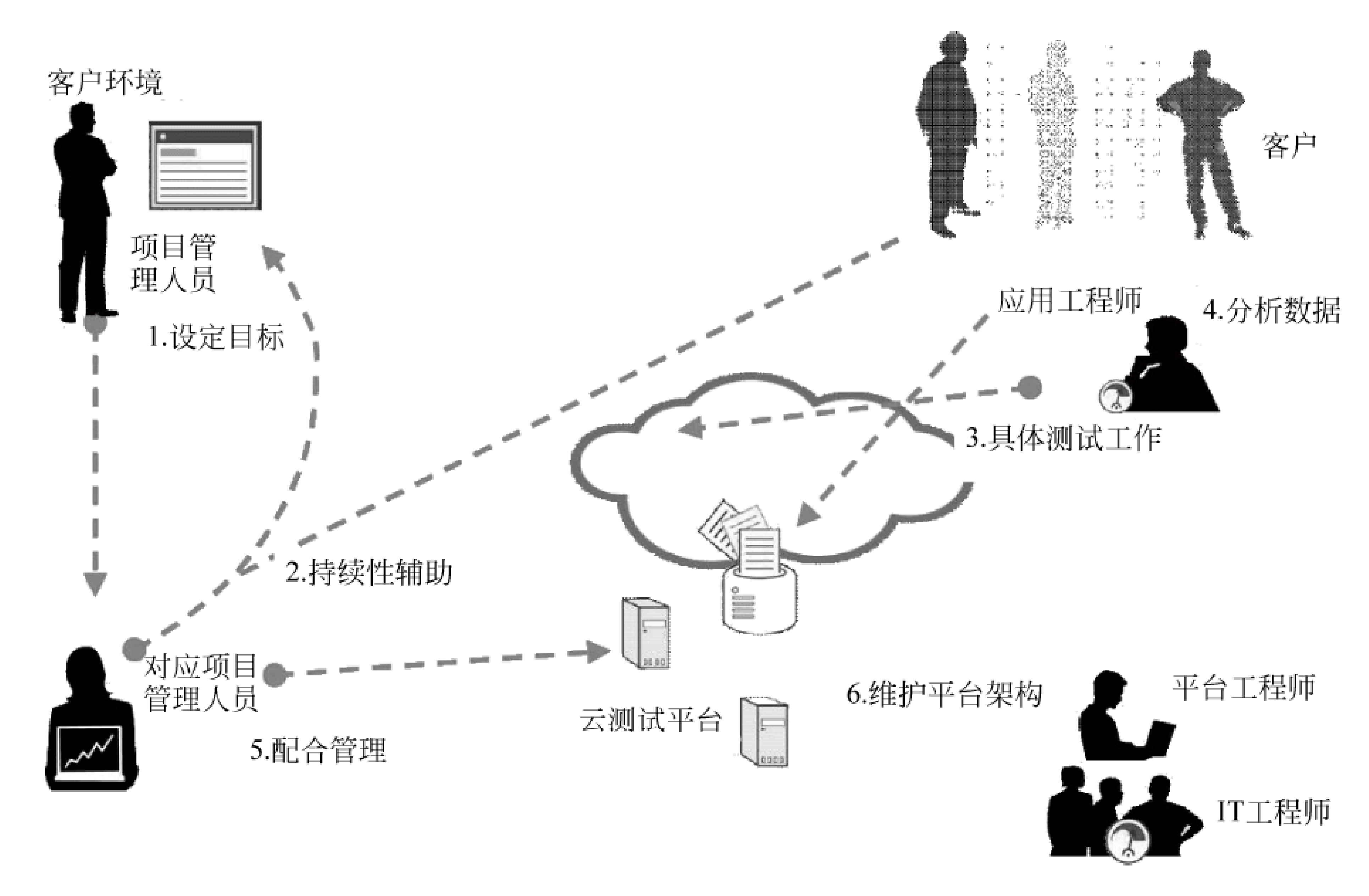


图 16-7 云测试平台应用流程图

目实施费和定期的服务费,即可通过互联网享用测试系统);④基础架构无关性(云用户不必再负担包括硬件设备、网络环境、操作系统、数据库、中间件等在内的基础结构,IT 工作人员以及诸如应用程序管理、监控、维护等操作性问题带来的成本);⑤高安全性(通过有效的技术措施保证云用户的数据获得高质量的安全性和保密性)。

2. 云测试平台中用到的云计算技术

在云测试中,要求测试人员对云计算技术有一定的理解才能更好地进行测试,以下对一些关键的云计算技术做出阐述。

1) 虚拟化技术

虚拟化是云计算的基础,能够实现资源的逻辑抽象和统一表示,其作用是将计算机资源整合成一个或者是分割成若干个操作环境,为上层的应用提供基础架构。虚拟化技术里面最核心的虚拟机的一些逻辑分类方式,是做云计算测试必备的知识。

2) 分布式存储

分布式存储,就是将数据分散存储在多台独立的设备上。分布式网络存储系统采用可扩展的系统结构,利用多台存储服务器分担存储负荷,利用位置服务器定位存储信息,它不但提高了系统的可靠性、可用性和存取效率,还易于扩展。这不是一个新技术,现在只是拿来为云计算提供服务。

3) 海量数据管理

云计算需要对分布的、海量的数据进行处理、分析,类似亚马逊、Google、淘宝等这样的互联网企业的发展,数据管理技术必须能够高效地管理大量的数据。云计算系统中的数据管理技术主要是 Google 的 BT(BigTable)数据管理技术和 Hadoop 团队开发的开源数据管理模块 HBase。

4) 云平台管理

云计算的资源规模庞大,服务器的数量众多,并分布在不同的地点,同时运行着数百种应用,如何有效地管理这些服务器,保证整个系统提供不间断的服务呢? 这是一个巨大的挑战。

云平台的管理涉及配置管理、生命周期管理、监控与诊断、质量管理等。对于云平台管理，应该积极关注管理的标准化、自动化和智能化。

习题

1. 云应用涉及哪些概念？它的工作原理是什么？具有什么样的特性？
2. 什么是云测试？云测试包括哪些测试内容？涉及哪些测试技术？

第17章 游戏软件测试

同其他行业一样，中国的电脑游戏产业经过前几年时而无序、时而高速的发展历程，如今一切都处于更良性的发展轨道上。随着人们对软件测试的重视，在游戏行业中软件测试部门也得到了比较多的支持，所以在游戏业中软件测试部门也逐渐发展起来。比如在对游戏软件测试的理解上，Microsoft 的游戏研发部门就是世界上最好的而且是最与众不同的，他们通常会让高学历、高素质的人去做质量保证方面的工作，回报当然也是丰厚且显而易见的。2006 年全球最佳游戏《Gears of War》，就是由 Microsoft 游戏部门开发的，只有 9 位测试人员就完成了对这款大作的测试。Microsoft 研发部门的技术实力和人员素质，以及对工作的组织都是令人惊讶的。在 Microsoft 的游戏测试部门有专用的工具开发工程师，如果某项测试工作可以用自动测试来完成，那么工具很快就可以被开发出来，这样就大大提高了软件测试的效率，使得游戏项目的质量非常之好。

而国内游戏软件测试行业目前非常复杂，有国际知名游戏公司在中国的研发中心内部的测试团队，有游戏代理商自己的测试团队，也有本土网络游戏研发中心的测试团队，还有现在正在崛起的中国游戏软件外包公司自己的测试团队，当然还有最近进入国内的专职测试服务公司。

目前，游戏产业迅猛发展，特别是近几年来，网络游戏成了网络最新的弄潮儿，从盛大之传奇般的崛起，吸引了无数公司的眼球。但由于随着玩家的品位的升高，代理费用的上升，单一的代理国外游戏的模式已经很难在国内立足，而有中国传统文化特色的网络游戏则在国内大受欢迎，比如剑侠情缘、大话西游等一些国内的经典之作已经进入了一流网游的阵营。与此同时，随着大家对网络游戏稳定性、可玩性要求的升高，网络游戏测试开始成为大家关注的话题。

游戏测试已经成为游戏产品开发的不可缺少的环节，然而随着游戏软件测试的发展，一些问题也随之浮出水面。我国的游戏软件测试特别是网络游戏软件测试技术还没有得到很好的研究或正处于初级阶段，游戏测试还没有一个完善的规范、标准、流程，其游戏测试大都是“小媳妇”角色，而且缺乏专业性，这些给我国网络游戏软件的发展带来一定的障碍。游戏的可玩性测试是一项新的测试需求，随着游戏的发展，这种需求还会不断增加。然而，游戏的可玩性测试的相关方法和技术还没有形成，游戏测试更多的是采用 β 测试来完成，即通过发布试用版游戏来达到检测可玩性的目的，而且随着对测试自动化的需求，传统的测试方法受到了很大的挑战。

17.1 游戏软件测试基本概念

游戏软件测试主要由两部分组成：一是传统的软件测试；二是游戏本身的测试。

游戏软件测试作为软件测试的一部分，具备了软件测试所有的一切共同的特性：①测试

2. 游戏的软件测试

在游戏软件测试过程中常遇到的错误有功能错误、赋值错误、检查错误、时间控制错误、打包错误、算法错误、文档错误及接口错误等。

(1) 功能错误：功能错误是一种影响游戏功能以及用户体验的错误，该错误可能是由提供这一功能的代码丢失或不正确造成的。例如命令角色向东，他却向西；没办法进行联网操作等。

(2) 赋值错误：当程序所使用的值被错误地初始化或设置，或者是当一个所需的参数值丢失时，出现的错误就被定义为赋值错误类型。例如，游戏任务开始，进入一个新关卡或一种游戏模式时，地图、角色属性或物件属性的值错误。

(3) 检查错误：当代码在被使用前不能适当地验证数据时，就产生了检查类型的错误。例如：在代码中用"＝"代替"＝＝"对两种值比较；边界比较，如使用"＜＝"代替"＜"等。

(4) 时间控制错误：时间控制错误与资源的共享、资源的实施管理相关。有些进程，如在硬盘上存储游戏信息，要给出开始时间或结束时间。这类操作在数据上执行，应完成对数据的操作后才能终止。通常为了友好，可以显示一个进度条或提示之类的信息。

(5) 由软件模块打包而导致的错误：这类的错误通常是由于配置游戏代码、变更游戏版本或安装打包等引起的。

(6) 算法错误：这种错误包括一些计算过程或选择结构中出现的有关时间复杂度或程序正确性的问题。算法可以视为得出一个数值(如 22)或实现一个结果(如关门)的过程。例如在一个填字游戏中会有的一些算法：点数、奖励和计数；完成一个回合或进入下一个关卡的标准；确定填字游戏目标的成功，如形成一个特殊的字或匹配一定数量的块；或者提供特殊的道具、奖励或游戏模式。

(7) 文档错误：文档错误发生在游戏已确定下来的数据素材中，其内容涉及文本、对话框、用户界面元素、帮助文本、指示说明、声音、视频、场景、关卡、环境对象、物件等。

(8) 接口错误：一个接口错误可以发生在任何信息被转移或交换的地方。在游戏代码中，当一个模块调用另一个模块的方式有误时，例如以错误的参数值调用函数、以错误的参数次序调用函数、遗漏参数去调用函数等，接口错误就发生了。

除上之外，游戏软件的逻辑部分大多会在后期进行，比如公测。是各种功能的数值调整。主要为游戏的世界定义一个平衡。除了初级的数值设定外，内部测试人员很少有能把一个功能测试千万遍的。于是有可能产生与游戏相关的逻辑问题。策划及相关测试人员注重的应该是这部分的测试原理及方法。

这部分问题的测试，同硬性问题一样，需要一定流程及要求。而具体流程只有根据具体游戏来决定。

17.2 游戏软件测试与游戏开发过程

17.2.1 游戏开发过程

要了解如何测试游戏必须了解如何做游戏，了解它的开发过程，才能真正地测试好游戏。表 17-1 给出了游戏软件项目与其他软件项目在开发上的一些区别。图 17-1 给出了游戏团队的人员组成，其中 GM 为游戏开发的管理人员，PM 为游戏产品的负责人。

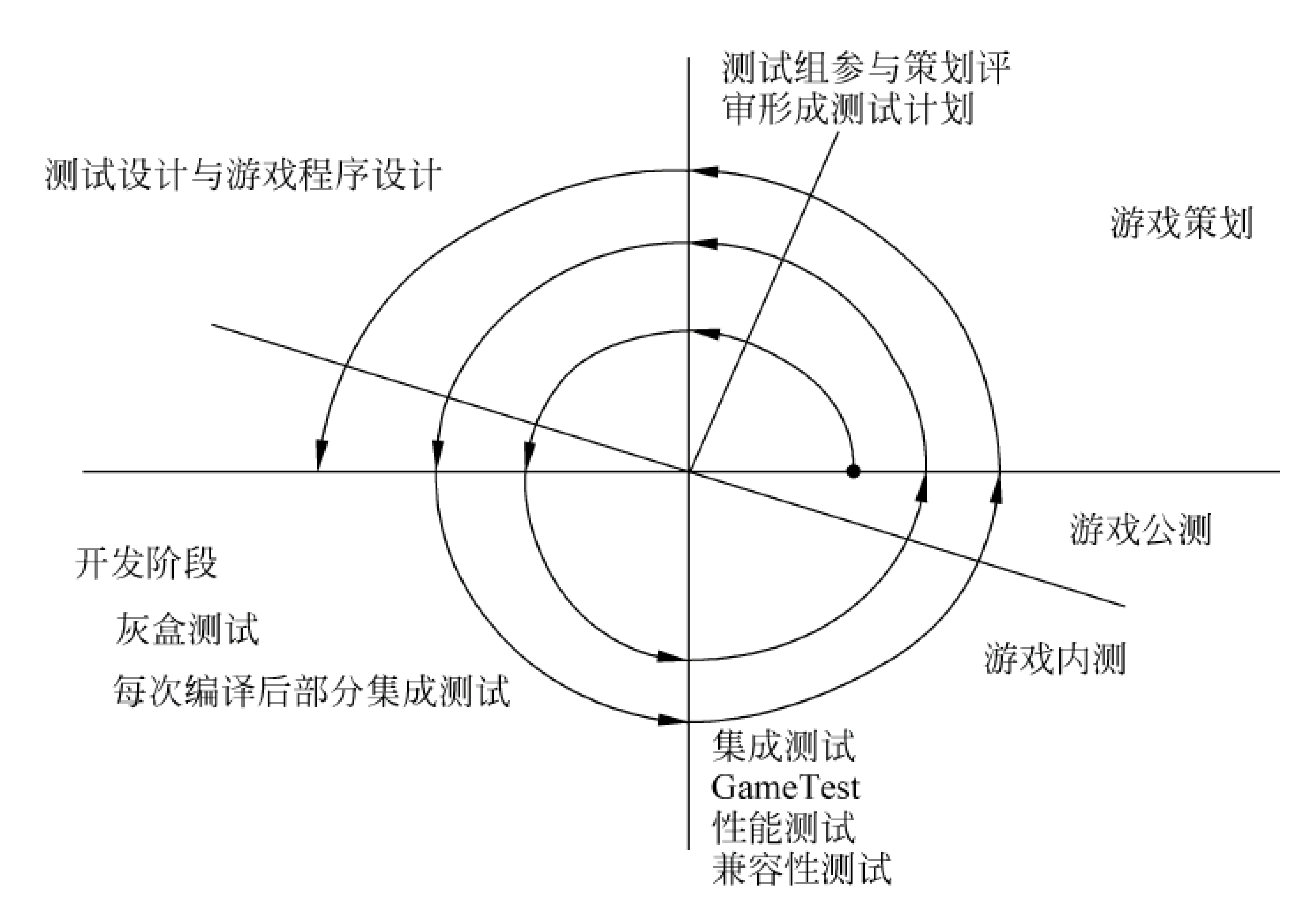


图 17-3 游戏迭代开发模型

确游戏的设计目标(包括风格、游戏玩家群)、游戏世界的组成,为后期的程序设计、美工设计、测试提出明确要求。由于开发是一个阶段的过程,所以测试与开发的结合就比较容易,从图 17-3 我们可以看到测试工作与游戏开发是同步进行的。在每一个开发阶段中,测试人员都有参与,他们深入了解系统的整体与大部分的技术细节,从而从很大程度上提高了他们对错误问题判断的准确性,并且可以有效地保证游戏系统的质量。

1. 游戏策划与测试计划

测试过程不可能在真空中进行。在策划评审中让测试人员参与进来,因为如果测试人员不了解游戏是由哪几个部分组成的,那么执行测试将非常的困难;同时测试计划可以明确测试目标、资源需求、进度安排等。通过测试计划,既可以让测试人员了解此次游戏测试中哪些是测试重点,又可以与产品开发小组进行交流。在企业软件开发中,测试计划书来源于需求说明文档,而在游戏开发过程中,测试计划的来源则是策划书。策划书包含了游戏定位、风格、故事情节、配置要求等。从策划书里可以了解游戏的组成、可玩性、平衡(经济与能力)与形式(单机版还是网络游戏)。而测试在这一阶段主要的事情就是通过策划书来制定详细的测试计划,这个测试计划主要分五个方面。

(1) 游戏程序本身的测试计划,比如任务系统、聊天、组队、地图等由程序来实现的功能测试计划。

(2) 游戏可玩性测试计划,比如经济平衡标准是否达到要求、各个门派技能平衡测试参数与方法、游戏风格的测试。

(3) 关于游戏性能测试的计划,比如客户端的要求、网络版对服务器性能的要求。

(4) 测试计划书中还要写明基本的测试方法、要设计的自动化工具的需求等,为后期的测试打下良好的基础。

(5) 由于测试人员参与策划评审,对游戏也有很深入的了解,会对策划提出自己的看法,包含可玩性、用户群、性能要求等,并形成对产品的风险评估分析报告。但这份报告不同于策划部门自己的风险分析报告,主要是从旁观者的角度对游戏本身的品质做充分的论证,从而更有效地对策划起到控制的作用。

2. 游戏设计与测试

设计阶段是设计系统的过程,是做测试用例设计的最好时机。很多组织要么根本不做测

试计划和测试设计,要么在即将开始执行测试之前才飞快地完成测试计划和设计。在这种情况下,测试只是验证了程序的正确性,而不是验证整个系统本该实现的东西。而我们的测试则会很明确,因为我们的测试计划已经写得很明确,需要测试哪些游戏系统。但是我们还需要了解系统的组成,而设计阶段则是设计系统的过程,所有的重要系统均是用统一建模语言 UML 状态图进行详细的描述,比如用户登录情况,如图 17-4 所示。

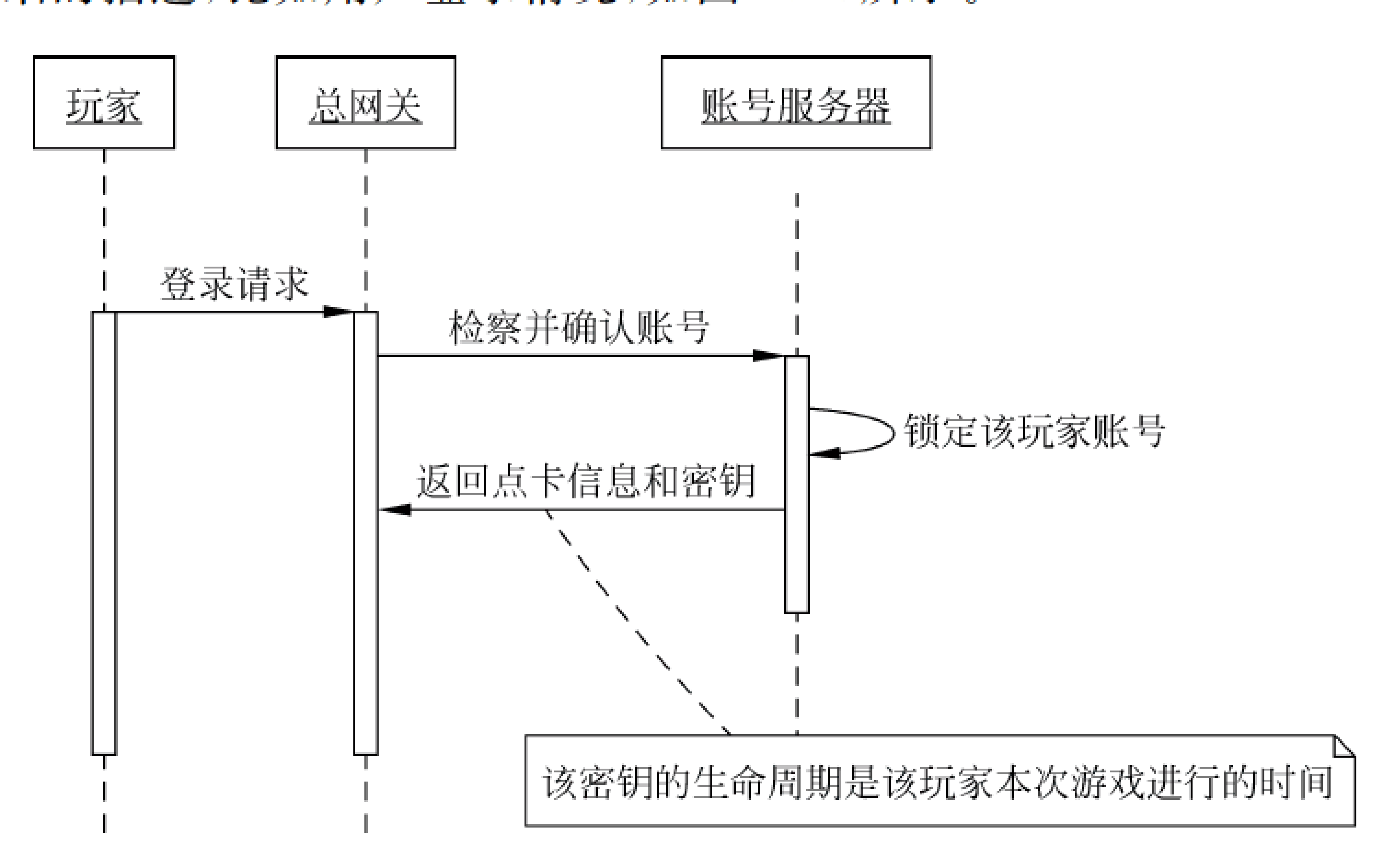


图 17-4　用户登录 UML 图

对于团队中的资深测试人员来说,要具备的一项基本素质就是可以针对 UML 的用例图、时序图、状态图来设计出重要系统的测试用例。只有重要系统的质量得到充分的测试,游戏程序的质量才可以得到充分的保证。比如图 17-4 就是一个用户登录游戏系统的时序图。从这里我们可以很明确地了解玩家是如何验证并登录系统的,在这个过程中要与哪些对象进行交互,这里是三个系统之间的交互——客户端(玩家部分)、网关、账号服务之间的一个时序变化关系。

为了能够完整地对这个流程进行测试,我们必须设计出可以覆盖整个流程的测试用例,并考虑其中可能的非法情况,因为这个时序图只是考虑了用户正常登录成功的情况,并没有考虑密码错误、通信失败等许多可能存在的情况。并形成完整的测试用例库,从而对登录系统的系统化测试做充分的准备。同时通过图 17-4,性能分析人员还可以分析出可能存在的性能瓶颈,比如这里可能有的瓶颈:总网关可以达到多少用户的并发,如果达不到,是否可以采用分布式部署或是支持负载平衡;三者之间的网络带宽的比例分配;账号服务器是否可以承载多个网关的连接请求;最大连接请求可以达到多少等。同时会针对这些风险做性能测试的设计,并提出自动化测试的需求,比如模拟玩家登录的压力工具,等等。

在设计评审时,测试人员的介入可以充分地对当前的系统构架发表自己的意见。由于测试人员的眼光是最苛刻的,并且有多年的测试经验,可以比较早地发现曾经出现的设计上的问题,例如用户登录测试,是客户端(玩家部分)、网关和账号服务三个系统之间的交互;为了能够完整地对这个流程进行测试,测试人员必须设计出可以覆盖整个流程的测试用例,并考虑其中可能的非法情况,如密码错误、通信失败等许多可能存在的情况;然后形成完整的测试用例库,从而对登录系统的系统化测试做充分准备;同时性能分析人员还要分析用户登录可能存在的性能瓶颈并设计测试用例,如总网关是否可以达到多少用户的并发,若达不到,是否可以采用分布式部署或是支持负载平衡,三个系统之间的网络带宽的比例分配,账号服务器是否可以承载多个网关的连接请求,最大的连接请求可以达到多少,等等。又例如玩家转换服务器测

试，在玩家转换服务器时是否作了事务的支持与数据的校验，在过去设计中由于没有事务支持与数据的校验从而导致玩家数据丢失，而这些风险可以在早期就规避掉。

上面所说的是对游戏程序本身的测试设计，对于游戏情节的测试则可以从策划获得。前期的策划阶段只是对游戏情节大方向上的描述，并没有针对某一个具体的情节进行设计，而进入设计阶段时，某个游戏情节逻辑已经完整地形成了，策划可以给出情节的详细设计说明书（又称为任务说明书），通过任务说明书我们可以设计出相应的测试用例，比如某一个门派的任务由哪些组成。我们还可以设计出完整的任务测试用例，从而保证测试可能最大化地覆盖所有的任务逻辑，如果是简单任务，还可以提出自动化需求，用软件自动完成。

3. 集成测试阶段

集成测试是对整个系统的测试。由于前期测试与开发的并行，集成测试已经基本完成，这时只需要对前期在设计阶段中设计的系统测试用例运行一下就可以了。我们主要的重心是集成测试中的兼容性测试，由于游戏测试的特殊性，对兼容性的要求特别高，所以我们采用了外部与内部同步进行的方式。内部我们有自己的平台实验室，搭建主流的软硬件测试环境，同时我们还通过一些专业的兼容性测试机构对我们的游戏软件做兼容性分析，让我们的游戏软件可以运行在更多的机器上。

4. 游戏可玩性测试

游戏可玩性测试也是非常重要的一块，主要包含 4 个方面：①游戏世界的搭建，包含聊天功能、交易系统、组队等可以让玩家在游戏世界交互的平台；②游戏世界事件的驱动，主要指任务；③游戏世界的竞争与平衡；④游戏世界的文化蕴涵，例如游戏的风格与体现。

这种测试主要体现在游戏可玩性方面，虽然策划时我们对可玩性作了一定的评估，但这是总体上的，一些具体的涉及某个数据的分析，比如 PK 参数的调整、技能的增加等一些增强可玩性的测试则需要职业玩家对它进行分析，这里我们主要通过三种方式来进行：①内部的测试人员。他们都是精选的职业玩家分析人员，对游戏有很深的认识，在内部测试时，他们要对上面的 4 点进行分析。②游戏媒体专业人员。利用外部游戏媒体专业人员对游戏作分析与介绍，既可以达到宣传的效果，又可以达到测试的目的，通常这种方式是比较好的。③外面的玩家。利用外部一定数量的玩家，对外围系统进行测试。他们是普通的玩家，却是我们最主要的目标，主要的来源是大中专院校的学生等，主要测试游戏的可玩性与易用性，发现一些外围的 Bug。

5. 游戏最后阶段的测试

游戏进入到最后阶段时，还要做内测、公测，有点像应用软件的 Beta 版的测试，让更多的人参与测试，测试大量玩家下的运行情况。可玩性测试是游戏最重要的一块，只有获得玩家的认同，我们才可能成功。

6. 性能测试与优化

最后要单独提一下的是性能优化。在单机版时代，对性能的要求并不是很高，但是在网络版时代，则是两个完全不同的概念，主要包含了以下几个方面：应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试。通常情况下，三方面有效、合理地结合，可以达到对系统性能全面的分析和瓶颈的预测。不过在测试过程中有这样一个原则，就是由于测试是在集成测试完成或接近完成时进行，要求测试的功能点能够走通，这时首先要进行优化的是数据库或是网络本身的配制，只有这样才可以规避改动程序的风险。同时性能的测试与优化是一个逐步完善的过程，需要前期做很多的工作，比如性能需求、测试工具等，不过由于前期工作的完善，这些都在前期完成了。

数据库优化的原则主要是这样的，首先是对索引进行优化，索引的优化不需要对表结构进

行任何改动,是最简单的一种,又不需要改动程序就可能提升性能若干倍,不过要注意的是索引不是万能的,若是无限增加就会对增删改造成很大的影响;其次是对表、视图、存储过程进行优化,不过在分析之前需要知道优化的目标,客户行为中哪些 SQL 是执行得最多的,所以我们必须借助一些 SQL 跟踪分析工具,例如 SQLProfile、SQLExpert 等工具,这样会迅速地定位问题。

关于网络的优化,这里我们所说的并不是针对网络本身的优化,而是对游戏本身的网络通信的优化,所以它是与程序的优化结合在一起的。首先也是发现问题,通过 Monitor 与 Sniff 先定位是什么应用占用了较多的网络流量,由于网络游戏的用户巨大,所以这也是一个重要的问题。对于程序的性能优化,最主要的是要找到运行时间最长的函数,只有优化它,性能才有大幅度的提升,具体的方法就不做详细描述了。

7. 兼容性测试

兼容性测试的目的是使游戏在不同的软件和硬件配置中正常运行,玩家手中 PC 的硬件和操作系统的种类与开发人员、测试人员使用的机器有一定的差别;玩家在自己 PC 上安装不同的软件,例如驱动程序、应用程序,而且在任何时候都可能运行许多不同的软件,这样可能造成一些无法预想到的情况,等等,这些都需要在游戏测试时考虑到。因此,在条件允许的情况下尽可能进行多种组合测试,比如操作系统的兼容性测试,要在当前主流的 Windows XP/7 等操作系统上测试游戏;硬件兼容性测试,要测试一些显卡是否与游戏存在冲突,某一型号的驱动程序是否与游戏存在冲突等。

17.3 网络游戏测试

网络游戏是指大型的多人在线角色扮演游戏,此类游戏与人们通常所玩的一般电子游戏所不同的是,它是人们通过互联网而进行的一种对抗式的电子游戏。在游戏中,你的对手不再是单一的由程序员编制的电子动画,还可以是藏在电子动画后面的人,即所谓的玩家。网络游戏的乐趣是人与人之间的对抗,而不仅仅是人与事先设置的各种程序的对抗,所以网络游戏比普通的电子游戏更具有生命力,更具有诱惑性。而且网络游戏世界从某种意义上说是另一个人类社会,人们在网络游戏世界中在被允许的范围内活动,比如说修炼、交流、合作、经商、欺诈、情感、冲突等。在制作网络游戏时反映这些行为的部分就是一个个完整的功能,所有这些功能组合在一起,就形成了网络虚拟世界。

在测试网络游戏时,需要考虑如下问题:

(1) 网络游戏的功能是否都实现了?

(2) 人们在进行操作时会如何做?比如说可能有多少种做法,这些做法应该有什么样的响应,哪些做法是被禁止的,在进行了被禁止的操作后应该有什么响应,等等。

关于网络游戏的测试,除了上面所讲的测试外,还有很重要的两种测试——网络游戏的平衡性测试和网络游戏的性能测试。

17.3.1 网络游戏的平衡性测试

1. 平衡性

所谓平衡性,在广义上来说包括:①玩家与玩家间的平衡性(技术水平的差距),主要体现在竞技类游戏和即时战略游戏中;②玩家与游戏规则间的平衡性(体现在可玩性),主要体现在如“大富翁”等休闲游戏中;③游戏内部的平衡性(包括各个系统的平衡),主要体现在社区

型多人在线游戏中。

我们可以把游戏看做一个多样化选择的集合。玩家在游戏中每走一步、每个阶段，都可以做出多种多样的选择来进入游戏的下一个阶段，来实现自己在游戏中的乐趣。但是如果当游戏进行到某一个阶段，那些多样的选择逐渐减少，甚至出现唯一的选择，而这时游戏仍然在进行中，那么显然这个游戏就失去了平衡。

而对于游戏中一个个单独的系统如经济系统等，平衡就是指系统对玩家的给予和系统从玩家那里回收的平衡。这个平衡就如上面所讲到的并不一定是指数值上的相等，而有可能是指在很长的一个时间段或者阶段，系统对玩家的给予和系统从玩家那里的回收是按照一定的比例存在的。这个比例就是整个系统平衡的关键。这也是游戏策划需要努力寻找并且在实践中不断调整的一个比例。

为什么要进行游戏的平衡性测试？平衡性测试一般遵循游戏性测试的原则，因为平衡性测试是游戏性测试的一个重要组成部分。

2. 平衡性测试

在进行平衡测试之前，一定要明确平衡的目标以及平衡的设定。如何让各种设定不偏离主题，明确世界背景及制定等级概念应该是首要的。尤其在一些角色等系统十分复杂的情况下。那种变态 ADD(加载)的规则，可由主角的 5～6 种基础属性影响到数十种战斗、非战斗技能。还可根据各种物品来修正这些数值。而无论如何，他们都有个明确的等级观念。从弱到普通，再到强，甚至到最强的龙。这是因为他们知道一个人，最强也不能强过龙。这样就给自己定下上限目标。

平衡性测试时首先不要去看玩家可选择的职业技能等是否足够多，都会获得什么强大的技能、体力等。先了解这个世界中各个种族之间的关系、职业的互补、各个角色的互相的关系，在整个世界中是什么位置，是否够合理，让常人可以以现实中的逻辑去衡量，这个角色在游戏中是否合理。之后才需要针对每个种族、每类职业、每个角色的平衡，进行一个一个角色的测试。

在这里定义的过程，正好与现实世界中相反。现实世界是总结出整体的平衡，而游戏世界则要定义平衡，再将世界整理成平衡的状态。

3. 等级测试

测试时同样要明确问题的严重等级，一个数值影响的事物越多，那它的严重等级越高。

现在的 MMRPG(多人在线角色扮演类游戏)整个属性结构，基本都类似树形结构，之间也有着一些交错的枝叶、力量等最基本的角色属性。这类属性会影响到其他属性，最终到达游戏的胜负、任务的完成等。而这些属性的等级自然也就十分明确。根为最高，枝叶最低，而修正树木永远不会从根开始。

力量，最基础的属性，结合自己的命中率、对方的敏捷性等，会影响物理攻击。同样也影响着可拿的武器。但如果这个人攻击力过高。那是谁的原因？是武器，还是角色的力量？需要修改哪一个？哪些角色的基础属性是最不能随便修改的？因此，还是武器吧。实在不行再从由属性引发的其他部分着手，如技能的熟练度等。因此，越是基础的部分，影响力越大，也最容易出错。

角色的基础属性是一切测试的根源，同样也是最不能随便更改的一类。更不应该因为某个问题而被指明要求更改。添加、删除任何一个属性，都会让之前的测试工作有 2/3 付之一炬，也许更多。

而对于各种武器，基本可以与角色测试分开。在角色属性有数十条的游戏中，武器更不会容易出现大的问题。

严重等级从高到低可分为角色、物品、技能。要修正这三大类属性，尽量在自己的范围内修正。不要妄想在其他级别动手，更别想在比自己高的级别里动手脚。而在这些属性里面同样还有各种属性，这就需要根据具体游戏进行划分测试了。虽然这里以属性举例，但任务也同样如此，相互关连的任务同样十分重要。只不过它们的变化较属性略少。

在现实世界中，是不可能通过走捷径获得某一种事物的，只有拼搏才能成功。游戏世界里是否也是这样？玩家的付出是否与获得成正比？或者说，在获得一个强大技能之前，给角色的锻炼是否足够？是否能够让他足够珍惜这一种技能或物品？这是游戏中较为关键的一部分，这多体现在任务上。时间、精力的消耗，是否足够让玩家在获得物品时有足够的满足感，以及对得起测试人员的劳动？

4. 游戏性测试

游戏性测试通常是作为功能测试的一种弥补而存在的。但是我们绝对不能忽略游戏性测试，因为在进行正常的游戏测试时，即使我们仔细地分析了策划文档，研究了测试计划，即使我们设计出了一份非常详细的测试用例书，甚至我们为了测试这个项目不惜分析一遍以前的案例，我们也依然有可能遗漏某些地方的问题，而这些地方大多数就是游戏性的问题。

游戏性测试和功能测试的最大区别在于，当我们没有被给予一个明确的目标来重现一个较大的缺陷时，我们需要带着所有的疑问进入游戏，在游戏的探索过程中来发现这些问题，因为游戏性测试更依赖于我们对游戏的理解和直觉。

17.3.2 魔兽世界的平衡性测试

为了更好地阐述平衡性测试，以下我们用一个具体的游戏——魔兽世界作为对象来展开测试。

1. 魔兽世界基本介绍

魔兽世界是建立在著名的魔兽故事基础上的一个大型多人在线角色扮演游戏。玩家在广阔的世界中探索、冒险、完成任务，其目的就是要成为魔兽世界中的英雄。魔兽世界可以让数千名玩家通过互联网在同一世界中进行交互。无论是一起冒险还是在史诗般的战斗中，玩家之间会建立友谊，组成联盟并且一起为了力量和荣耀而共同与敌人战斗。

魔兽世界在玩家群体之中添加了很多更具游戏性和互动性的系统，比如拍卖行等。需要提到的是，魔兽世界的经济系统平衡性做得很好，但是到了游戏的后期仍然存在很多平衡性的问题，比如由于高级战利品的价格较高而系统又缺少新的玩法来有效地回收这些过剩的金钱，导致玩家金币积累过多而造成平衡失调，最直接的表现如最初的魔兽金币的线下交易每个几乎需要 2 元，而现在却只需要几分钱。这就是为什么到了后期为了避免游戏中通货膨胀，而需要进行像安琪拉开门以及引进新的种族和开放外域空间的世界性事件，通过更多的玩法来回收玩家的金钱。

在魔兽世界中，玩家从经济系统获得金钱只有两种途径：①打怪，得到一定数量的金钱或是装备，战利品可卖给 npc 商店换取金钱；②任务奖励，玩家通过做任务获得金钱，或者装备的奖励。

我们把运用生活技能进行采集等获得奖励的途径看做一种变相打怪行为，因此将其归为第一类途径。而系统从玩家处回收包括学习技能、装备修理、购买消耗品以及飞行费用等。

2. 基于魔兽世界所进行的游戏初期的平衡性测试

1）分析文档

明确魔兽世界的基本构架，理解策划人员对平衡性的要求，也许有的需要保持平衡性的地

方有明确的说明，但是有的更细节的方面就没有明确的策划文档依据。尝试理解整个游戏，才能够具有足够敏锐的嗅觉来发现问题。要知道测试人员实际上并不清楚哪里会出现问题，但是遍历游戏的所有玩法是不可能的，所以测试人员要通过设计有效的测试用例来完成。

设计测试用例和执行用例：根据策划文档的分析结果，来设计测试用例。对于平衡性测试来说，寻找一贯占优势的方法或者一贯不被采用的方法来进行游戏测试，并且将这两种情况归结到典型的不平衡当中，作为边界值来使用。比如测试人员发现大多数玩家热衷于 farm 而不去理会那些很有吸引力且回报颇丰的任务。对于这种情况，测试人员在设计用例的时候一定涉及，但是测试人员不能在这个时候就简单地认为这是不平衡的，因为测试人员并不知道这些情景下所得到的结果是不是像预期的那样，所以一切都要实施测试之后才能得出结论。需要注意的是，在测试的过程中，测试人员要将系统的回收（如修理装备、购买消耗品）从最后的玩家获得中扣除，最后得到玩家的纯收益，一般都用金钱来衡量。

在选择测试用例的时候，要明确测试目的。测试人员是要对经济系统进行测试，所以要尽可能地排除其他事件对测试过程的影响，如角色的死亡以及在两块大陆或者多个飞行点之间的来回穿梭的时间等。也可能需要测试人员进行反复测试来得出真实有效的测试结果，最后再来对照策划文档进行测试结果的分析，如在一定的时间或者阶段，系统的产出和回收的关系是怎样，是不是符合策划的要求。

需要注意的问题是等价类的划分：准确地划分等价类，可以节省大量的测试时间。下面是针对魔兽世界进行等价类划分的一个例子：1～10 级是测试开始实施的阶段，需要进行测试；10 级以后由于开启了天赋系统，会对角色的劳动效率造成一定的影响，所以 11～20 级也要进行测试，而 31～40 级可以被认为是 20～30 级的一个等价类，因为在其间没有转职或引入新的系统；41～45 级则需要进行测试，因为在 45 级涉及一次固定资产的投入——购买坐骑，以此类推。同样，不同种族的相同职业，也可以看做等价类。如果可以实现从任意等级开始实施测试的话，我们也可以覆盖更多的等价类，得到多个不同等级阶段的测试结果，要知道越多的数据将会使测试结果越趋于精确。

2）设计用例

建立人族战士角色，从 1 级开始进行游戏，通过打怪、做任务或两者结合进行升级。

（1）通过打同等级怪进行升级，在其到达 10 级的时候，分析角色获得的金钱，并与策划文档进行比较，看是否达到策划的要求；

（2）只通过做任务进行升级，在其到达 10 级时，分析角色获得的金钱，并与策划文档进行比较，看是否达到平衡；

（3）通过正常方式升级（即做任务同时有目的打 farm 怪物），在其到达 10 级时，分析角色获得的金钱，并与策划进行比较。

以上都是很多玩家根据自己的兴趣所在而选择的游戏方法，但是测试人员要考虑到有些玩家会使用其他方法来提高劳动效率，比如购买大量的药品来辅助杀怪，或者是在可能的情况下，使用药品来击败高级怪物以及完成某些团队任务（主要是杀死高难度怪物的任务）。

17.3.3 网络游戏的性能测试

目前网络游戏主要分为传统的 C/S 架构的网络游戏、B/S 架构的网络游戏和 WAP（Wireless Application Protocol，无线应用协议）网络游戏。这三种网络游戏的性能测试是各不相同的。

1. 传统的 C/S 架构的网络游戏

传统的 C/S 架构的网络游戏历史最悠久，也是目前最主流的网络游戏类型。这类游戏需

要用户下载客户端，然后通过客户端来访问服务器进行登录和游戏。

这类游戏的性能测试方法大体有三种。

(1) 目前较常规的做法就是自主研发一个机器人程序，模拟玩家登录与游戏。这种方法的好处一是操作方便，对执行性能测试的人员无要求；二是能够较真实地模拟出玩家的部分操作。但是缺点也不少，如对开发人员要求较高，因为不仅需要模拟用户访问服务器，还需要收集多种数据，并且将数据进行实时计算等，成本较大，而且也不易维护。除此之外，机器人发生问题的时候，维护起来也不够方便。在复杂架构下不利于判断瓶颈所在位置。最重要的是一旦机器人开发进度拖延或者出现致命 Bug，性能测试将无法进行。

(2) 使用现成的性能测试工具来进行性能测试。可以使用工具来模拟用户与服务器交互的底层协议来进行测试。这种方法的优点是灵活方便、易于维护，开发成本小；增加删除性能点极其容易，发生问题也能立即维护；开发成本相对于机器人来说要少很多，并可以较容易地判断出性能瓶颈所在的位置。这种方式的缺点也不少，如对性能测试人员的要求比较高，需要根据用例来编写模拟用户与服务器之间的协议交互脚本；对于模拟真实性方面也比机器人程序差些。

(3) 使用最广泛，且与上面两条不冲突，即进行封测、内测、公测等开放性测试方法是最真实、有效的方法。让广大的玩家在测试服务器中进行游戏，在帮助游戏公司找到游戏中的 Bug 的同时，也对服务器的压力进行真实的测试。但这种方法难以控制。

2. B/S 架构的网络游戏

B/S 架构的网络游戏现在越来越流行，现在越来越多的人喜欢上了这种类型的网络游戏。它没有传统的 C/S 架构的网络游戏那种炫目的效果、唯美的画面，也没有传统网络游戏那种直观的人物动作，但是却吸引了越来越多的上班族去玩它。因为它有着传统的 C/S 架构的网络游戏不具备的优势，那就是方便、简单、要求低。只要可以上网，只要有浏览器，就可以进行游戏。无须下载客户端，无须担心机器配置不够，也无须长时间去投入，就可以享受到网络游戏的乐趣。

这类游戏的性能测试方法大体有两种：①使用工具来模拟用户访问，这个和其他的 B/S 架构的软件产品一样。通过各种工具、各种协议来模拟用户访问服务器，与服务器进行交互。②和传统的 C/S 架构的网络游戏一样，它也有封测、内测、公测等活动，让广大的玩家为游戏公司进行性能测试。

3. WAP 网络游戏

WAP 网络游戏现在也是越来越多了。这类游戏的性能测试方法大体有两种：①使用模拟器在电脑上模拟 WAP 环境，然后使用工具来进行性能测试。使用的协议可以是 WAP，也可以是 SOAP 等其他协议。②与其他两种网络游戏一样，都少不了开放性测试这个环节。

17.3.4 网络游戏的压力测试

对网络游戏进行压力测试主要考虑如下几个方面：

(1) 服务器方面考虑每台服务器最大的承压能力，在允许范围能保证多少人在线？如果超过这个正常数值会有什么应对办法？

(2) 考虑服务器地图切换时同一地图场景中在线人数过多的应对办法。

(3) 测试游戏正常运行所需要的最低网络带宽数值，并且考虑电信与网通互访的问题。(如果采用分线路服务器另说)

(4) 数据库考虑表是否建立得足够细分了，是否符合范式，是否按照数据库的建模语言来

做的数据库架构。

（5）服务器的最大承受压力这个还很粗，需要细化。比如最多有多少个fb（副本）存在，同时可以多少人连接银行、存取物品，等等。关键是你需要先分析在整个游戏中的不同操作，哪些是常用的，哪些不常用，常用的最大并发量是多少，然后模拟尝试确认系统的最大负载能力。

17.4　手机游戏测试

除了单机版游戏、游戏机上的游戏以及网络游戏外，手机上的游戏功能也越来越多地被人们所使用。手机游戏软件作为应用软件，其测试内容与一般的应用软件没有多大差别。

17.4.1　手机游戏软件的测试内容

手机游戏软件的测试内容分为8个方面：①用户文档（重点评测文档的完整性、正确性、一致性、易理解程度和易浏览程度）；②功能性（重点检测软件的安装与卸载、功能表现、正确性和一致性等）；③可靠性（重点检测软件的成熟性、容错性和易恢复性）；④易用性（重点检测软件的易理解性、易浏览性和易操作性）；⑤维护性（重点检测软件的易分析性和易改变性）；⑥可移植性（重点检测软件的适应性和兼容性）；⑦效率（重点考虑时间特性和资源特性）；⑧中文特性（检测重点是中文显示、汉化程度和编码支持程度）。

17.4.2　手机游戏软件测试的自身特性

事实上手机游戏软件测试与一般的软件或游戏软件测试有很大的区别和自身的特性，下面我们来重点讨论这些特性。

1. 功能性测试

手机的功能测试会因手机所处的使用条件而对游戏产生影响，例如：

（1）游戏运行中接听电话、短信后，能否返回到中断的游戏画面继续游戏。

（2）游戏设置中是否可以关闭声音、振动功能。

（3）游戏菜单中有无详细的操作帮助说明。

（4）棋牌益智类游戏能否积分上传等。

2. 手机游戏娱乐性内容的评价

手机游戏娱乐性内容的评价包括画面评价、游戏性评价和操作性评价三部分。

1）手机游戏画面评价

（1）游戏背景——游戏背景层次是否丰富鲜明、制作精细、像色数高、与前景用色对比明显；

（2）游戏前景——游戏场景中前景数量是否较多，造型丰富各有特点，细节刻画丰富，颜色丰富，与游戏内容相符；

（3）人物和物品造型——角色的肢体细节及物品造型是否丰富、比例正常、色彩艳丽；

（4）人物动作或物体运动状态——人物动作攻击、移动动作是否姿势丰富、流畅连贯、制作精细、无跳跃感；

（5）游戏特效——游戏中出现特效数量多少，效果是否细腻等。

2）游戏性评价

（1）关卡设计——评价故事设定是否完整，游戏中任务、谜题安排是否合理，场景设计问题，关卡设计等；

2）暴力度指标

（1）游戏内容健康，无明显暴力场景或过分暴力的战斗设计可得 1 分；

（2）轻度暴力，战斗过程血腥场景较少，无对角色身体的明显暴力的可得 2 分；

（3）游戏在战斗过程中出现将角色暴尸、尸体肢解等场景，违背人性范畴的得 3 分。

3）游戏定级条件

（1）适合全年龄段，各个项目的得分必须全部为 1 分，即适合所有未成年人，其中只包含最小程度的暴力、搞笑的恶作剧或者粗话。

（2）初中生年龄段以上，可以有暴力内容、轻度的粗话、极其少量的色情题目。暴力、色情、恐惧度均为 1 分，社会道德、文化内涵度拥有至少 1 项或最多 2 项 2 分。动态指标中非法程序（外挂）、PK 行为、社会责任感等程度至少拥有 1 项或最多 3 项 2 分。

（3）高中生年龄段以上，有比初中生年龄段层次更多的暴力及色情内容。静态指标中除色情度为 1 分外，其余 4 个指标中，至少 3 个最多 4 个达到 2 分。动态指标中，游戏形象宣传度为 1 分，其余 6 个指标，至少 5 个最多 6 个达到 2 分。

（4）18 岁年龄段以上，只面对 18 岁以上人士，不应该对未成年人推广发行。静态指标中任意一项指标达到 3 分，且达到 3 分的指标少于 4 项。动态指标任意一项指标达到 3 分。

（5）危险级，提请主管部门对其关注。有 4 项以上指标达到 3 分或者其服务过程中出现任何针对玩家的恶性突发性事件。

习题

1. 游戏软件测试相对传统软件测试而言具有哪些特点？一般包括哪些测试内容？
2. 游戏软件中常出现的错误有哪些？为什么会出现这些错误？
3. 简述游戏软件测试过程与游戏软件开发过程的关系。
4. 网络游戏中的测试重点有哪些？为什么？
5. 什么是网络游戏的平衡性测试？
6. 手机游戏测试包括哪些测试内容？对于这些测试内容，要重点突出哪些测试要点？



嵌入式软件测试

在讲嵌入式软件之前先介绍一下嵌入式系统的概念。嵌入式系统是计算机硬件和软件结合起来构成的一个专门的计算装置,用来完成特定的功能或任务。它是一个大系统或大的电子设备中的一部分,工作在一个与外界发生交互并受到时间约束的环境中,在没有人干预的情况下进行实时控制。

嵌入式系统由嵌入式硬件与嵌入式软件组成。硬件以芯片、模板、组件、控制器形式嵌入到设备内部,软件是实时多任务 OS 和各种专用软件,一般固化在 ROM 或闪存中。

一般嵌入式系统的软硬件可剪裁,以适用于对功能、体积、成本、可靠性、功耗有严格要求的计算机系统。嵌入式计算系统在硬件和软件的组成结构方面不像通用计算机那样有分明的层次,但它又是五脏俱全。

嵌入式系统主要用于各种信号处理与控制。在国防、国民经济及社会生活各领域被普遍采用,可用于企业、军队、办公室、实验室以及个人家庭等各种场所。嵌入式系统作为数字化电子信息产品的核心,凝聚了信息技术发展的最新成果,电子产品升级换代必须采用嵌入式系统。而芯片技术、软件技术、通信网络技术等嵌入式系统关键技术的新进展,也推动着嵌入式系统升级换代,如下。

(1) 军用:各种武器控制(火炮控制、导弹控制、智能炸弹制导引爆装置)、坦克、舰艇、轰炸机等陆海空各种军用电子装备,雷达、电子对抗军事通信装备,野战指挥作战用各种专用设备等。

(2) 家用:各种信息家电产品,如数字电视机、机顶盒、数码相机、DVD 音响设备、可视电话、家庭网络设备、洗衣机、电冰箱、智能手机、智能玩具等。

(3) 工业用:各种智能测量仪表、数控装置、可编程控制器、控制机、分布式控制系统、现场总线仪表及控制系统、工业机器人、机电一体化机械设备、汽车电子设备等。

(4) 商用:各类收款机、POS 系统、电子秤、条形码阅读机、商用终端、银行点钞机、IC 卡输入设备、取款机、自动柜员机、自动服务终端、防盗系统、各种银行专业外围设备。

(5) 办公用:复印机、打印机、传真机、扫描仪、激光照排系统、安全监控设备、手机、寻呼机、个人数字助理(PDA)、变频空调设备、通信终端、程控交换机、网络设备、录音录像及电视会议设备、数字音频广播系统等。

(6) 医用电子设备:各种医疗电子仪器,如 X 光机、超声诊断仪、计算机断层成像系统、心脏起搏器、监护仪、辅助诊断系统、专家系统等。

在上述应用中,微处理器、微控制器、DSP 芯片级嵌入式系统以及嵌入式软件是计算机、通信、仪器仪表等各类电子信息产品的核心。

现在越来越多的电子产品厂商采取贴牌生产 OEM 的方式把硬件制造外包出去,产品个性化竞争更多地体现在嵌入式软件的开发上,软件工程师扮演的角色越来越重要。

IDC 调查显示：在典型产品开发项目的全部人工费用中，软件工程人员的费用在 20 世纪 90 年代初期到中期约为 55%，如今已经达到 75%。发展嵌入式软件工程和技术是各相关产业腾飞的一个重要突破口。

18.1　嵌入式软件测试概念

18.1.1　嵌入式软件开发及应用特点

1. 嵌入式软件开发方法

一般采用典型的“宿主机/目标机”交叉方式，即利用宿主机上丰富的资源及良好的开发环境，通过串口或网络等将交叉编译生成的目标代码传输并装载到目标机，用调试器在监控程序或实时内核/OS 的支持下进行实时分析、测试和调试。目标机在特定的环境（如分布式环境）下运行。

通常嵌入式软件开发的最小编程环境主要是交叉编译、交叉调试器、宿主机和目标机间的通信工具、目标代码装载工具、目标机内驻监控程序或实时操作系统等。其中，交叉调试器在其中占有很重要的地位。

2. 嵌入式软件应用特点

嵌入式应用是一个特殊过程，具有软硬结合、资源约束及对外交互等特点。

（1）实时性，内存不丰富，I/O 通道少，开发工具昂贵，并且与硬件紧密相关，CPU 种类繁多，等等。

（2）它经常是被硬件体系结构、软件体系结构、操作系统特性、应用需求、编程语言及开发和调试环境的变化所驱动的。

（3）嵌入式应用与通常说的计算机应用相比，既要满足功能需求，还要满足性能需求，甚至性能需求放在第一位。

（4）嵌入式系统开发和通常意义上的计算机应用软件开发有很大不同。它不但要考虑软件设计，同时还要考虑硬件设计；不但要考虑功能设计，而且要考虑性能设计。

（5）最为重要的是按照设计思想开发出的嵌入式系统必须进行功能和性能验证。

（6）事实上，嵌入式系统开发的最大问题是设计、环境建立和验证（不是编程）。

18.1.2　嵌入式软件测试问题及传统测试方法

软件测试在整个软件开发过程中处于非常重要的地位，其测试费用占项目总费用的 25% 以上，对于嵌入式软件来说花费更大。可以说嵌入式软件是最难测试的一种软件。

1. 嵌入式软件测试问题

嵌入式软件测试同人们通常使用的传统软件测试相比有较大的差别，除了要考虑和运用传统的测试技术外，还要考虑与时间和硬件密切相关的测试技术运用，例如，对外部事件响应的测试问题。

在对嵌入式软件进行测试的过程中经常要用到各种技术，如嵌入式软件的分析技术或嵌入式软件的调试技术，因为实时嵌入式系统最大的特点是具有一组动态属性，即中断处理和上下文切换、响应时间、数据传输率和吞吐量、资源分配和优先级处理、任务同步及任务通信等。所有这些性能属性可以很容易地说清楚，但要测试或验证它们（特别是时间确认）是很困难的。对实时嵌入式系统进行分析需建模、仿真和数学计算。

软件调试在软件测试之后进行，用以定位和排除错误。对于嵌入式应用，无论是测试还是

调试,有效的方法仍是借助硬件仿真或软件模拟的手段来进行软件的测试和调试。

硬件仿真一般由硬件和软件构成,硬件提供低级的监控、控制和保护功能,包含仿真控制的处理器和所有的存储外围设备接口、通信口和在线仿真所需的硬件。而在仿真器里的软件提供状态和控制功能以及与宿主机的通信。在宿主机上工作的软件提供操作仿真器的用户接口;软件仿真通过数字化的形式仿真嵌入式软件的运行环境,包含支撑嵌入式应用的 CPU 的虚拟目标机、支撑嵌入式软件工作的外围硬件(元器件、电路、传感及外部设备等)的数字仿真手段。硬件仿真与软件仿真相比,主要优点是嵌入式软件是在真实的 CPU 上运行,它的不足是很难构造一个完整的嵌入式应用环境,很难支持嵌入式软件的先期开发、测试和调试。

2. 嵌入式软件的传统测试方法

嵌入式软件开发采用"宿主机/目标机"交叉方式,相应的测试也为 host-target 测试或 cross-testing。这是因为所有测试放在目标平台上会有很多不利的因素:①测试软件可能会造成与开发者争夺时间的瓶颈,要避免它就只有提供更多的目标环境;②目标环境可能还不可用;③比起主机平台环境,目标环境通常是不精密的和不方便的;④提供给开发者的目标环境和联合开发环境通常是很昂贵的;⑤开发和测试工作可能会妨碍目标环境已存在持续的应用。

交叉测试环境下应关注的问题:①测试需要多少人员(单元测试、软件集成、系统测试)?②多少软件应该测试,测试会花费多长时间?③在主机环境和目标环境有哪些测试软件工具,价格怎样,是否适合?④多少目标环境可以提供给开发者,什么时候提供?⑤主机和目标机之间的连接怎样?⑥被测软件下载到目标机有多快?⑦使用主机与目标环境之间有什么限制(如软件安全标准)?等等。管理者在进行嵌入式软件测试时都应深入考虑以上问题,结合自身实际情况,选定合理的测试策略和测试方案。

18.1.3 嵌入式软件测试策略及测试流程

嵌入式软件的测试与一般软件的测试不一样,它需额外考虑时间和硬件的影响和问题。对于硬件一般是采用专门的测试仪器进行测试,而对于软件,特别是实时嵌入式软件,则需要有关的测试技术和测试工具支持,需要采取特定的测试策略。

测试策略或测试技术指的是软件测试的专门途径,以及提供能够更加有效地运用这些途径的特定技术。包括覆盖测试、功能测试、性能测试、可靠性测试及回归测试等,用在软件开发过程中的不同阶段,如开发方的内部测试(单元测试、基于代码的覆盖测试或白盒测试)、第三方的验证和确认测试(功能测试或黑盒测试、性能测试)和维护中的修改和升级测试(回归测试)等。在测试过程中理想的测试工具希望它所支持的测试策略和测试技术越多越好,如覆盖工具、功能验证工具、内存分析工具、性能分析工具、基于 GUI 客户/服务方式的测试工具以及支持回归测试的测试自动化工具等。

1. 嵌入式软件测试各个阶段的通用策略

1) 单元测试

所有单元级测试都可以在主机环境上模拟目标环境进行,除非少数情况,特别是具体指定了单元测试必须在目标环境进行。

2) 集成测试

软件集成也可在主机环境上完成,在主机平台上模拟目标环境运行。当然在目标环境上重复测试也是必需的,在此级别上的确认测试将确定一些环境上的问题,比如内存定位和分配上的一些错误。

在主机环境上的集成测试的使用，依赖于目标系统的具体功能有多少。有些嵌入式系统与目标环境耦合得非常紧密，若在主机环境做集成是不切实际的。

一个大型软件的开发可以分成几个级别的集成。低级别的软件集成在主机平台上完成有很大优势，越往后的集成越依赖于目标环境。

3）系统测试和确认测试

所有的系统测试和确认测试必须在目标环境下执行。当然在主机上开发和执行系统测试，然后移植到目标环境重复执行是很方便的。对目标系统的依赖性会妨碍将主机环境上的系统测试移植到目标系统上，况且只有少数开发者会参与系统测试，所以有时放弃在主机环境上执行系统测试可能更方便。

确认测试最终的实施舞台必须在目标环境中，系统的确认必须在真实系统之下测试，而不能在主机环境下模拟。这关系到嵌入式软件的最终使用。

确认测试也包括恢复测试、安全测试、强度测试、性能测试。

另外，我们一定意识到好的交叉测试策略能提高嵌入式软件测试的水平和效率。

2. 嵌入式软件测试流程

在讲述嵌入式软件测试流程之前，先以嵌入式软件测试的“白盒”测试工具 CodeTest 为例，介绍一下嵌入式“白盒”测试的插桩概念（参见图 18-1）。

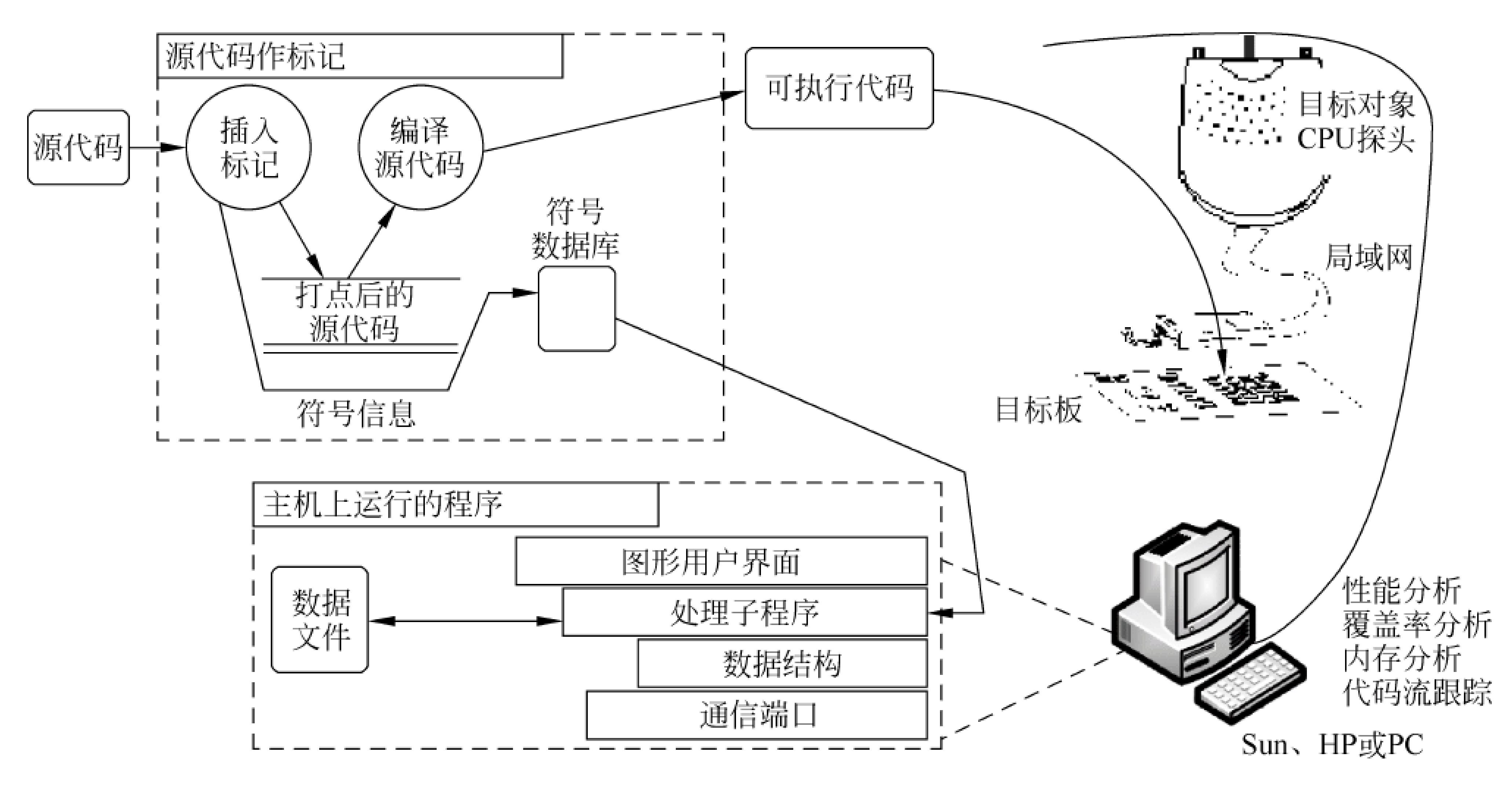


图 18-1 CodeTest 的插桩流程

程序员编写的源代码首先会通过 CodeTest 的编译驱动器调用相应的编译器进行预编译，然后 CodeTest 的插桩器（源代码分析程序）对预编译好的源代码进行自动插桩，即在需要插桩的关键位置写入一条赋值语句（如 amc_ctrt＝0x74100009），并把插入的标记送入一个数据库文件中生成一个符号数据库暂存起来，以备以后分析时调用。

然后，CodeTest 的编译驱动器又会调用原编译器，对插桩后的代码进行编译，生成可执行目标代码送到目标板上运行。当程序在目标系统运行到插桩点的位置时，目标板的控制总线和地址总线上会出现相应的控制信号和地址信号。当 CodeTest 的辅助硬件（信号捕获探头）从控制总线和地址总线上监视到符合以上条件的信号时，CodeTest 会主动地从数据总线上把数据捕获回来，送到 CodeTest 的内存中暂存并对这些数据进行预处理，然后将预处理后的数据通过局域网送到工作平台上。

通过与前面生成的符号数据库中的数据进行比较，我们就此得知当前程序的运行状态，借此完成对嵌入式软件的覆盖率分析。

嵌入式软件测试的一般流程：①使用测试工具的插桩功能（主机环境）执行静态测试分析，并且为动态覆盖测试准备好一插桩好的软件代码；②使用源码在主机环境执行功能测试，修正软件的错误和测试脚本中的错误；③使用插桩后的软件代码执行覆盖率测试，添加测试用例或修正软件的错误，保证达到所要求的覆盖率目标；④在目标环境下重复②，确认软件在目标环境中执行测试的正确性；⑤若测试需要达到极端的完整性，最好在目标系统上重复③，确定软件的覆盖率没有改变。

通常在主机环境执行多数的测试，只是在最终确定测试结果和最后的系统测试时才移植到目标环境，这样可以避免发生访问目标系统资源上的瓶颈，也可以减少在昂贵资源（如在线仿真器）上的费用。

另外，若目标系统的硬件由于某种原因而不能使用时，最后的确认测试可以推迟直到目标硬件可用，这为嵌入式软件的开发测试提供了弹性。

设计软件的可移植性是成功进行交叉的先决条件，它通常可以提高软件的质量，并且对软件的维护大有益处。

很多测试工具都可以通过各自的方式提供测试在主机与目标机之间的移植，从而使嵌入式软件的测试得以方便地执行。

18.2 嵌入式软件测试工具

按照传统的软件测试大类划分，嵌入式软件测试也分为静态测试和动态测试。其中动态测试分为“白盒”测试和“黑盒”测试。下面我们基于这种方法介绍一些典型的嵌入式软件测试工具。

18.2.1 嵌入式软件测试的典型工具

1. 嵌入式“白盒”测试工具

“白盒”测试以源代码为测试对象，除对软件进行通常的结构分析和质量度量等静态分析外，主要进行动态测试。

IBM Rational 的 Logiscope TestChecher 和 IBM Rational 的 Test Realtime，通过串口、以太网口与被测软件运行的目标机进行连接，在对被测软件进行插桩后下载到目标机上运行，进行准实时的或事后的分析。

美国 Freesacle 公司的 CodeTest 与被测目标机通过总线或飞线方式进行连接，将被测软件进行插桩，当被测软件在目标机上运行时对其进行实时的监测。

代表了当今软件自动化测试领域最高水平的高级语言单元测试工具 VectorCAST 支持单元的“白盒”测试。

国内有通过总线监听手段进行覆盖测试的，但当 CPU 采用指令预取方式工作时这种方式就有问题了。

2. 嵌入式“黑盒”测试工具

“黑盒”测试将嵌入式软件当做一个黑盒子，只关注系统的输入输出。目前的测试做法是以硬件方式将被测系统的输入/输出端口用硬件对接相连，使用实时处理机和宿主机对被测系统进行激励和输入，实施驱动，然后获取输出结果进行分析，进行开环或闭环测试。

这种方法的优点是实时性强，缺点是这种测试实际上是对整个被测系统的测试，是一种确认性测试。如发生问题，不知道是硬件还是软件发生的问题，抑或是软硬件耦合发生的问题。如果目标机未设计制造出或无法得到，这种测试就无法进行。

代表性的公司和产品是德国 Tech SAT 的 ADS—2 系统，价格比较昂贵。国内的是北航的 GESTE 嵌入式系统测试环境。

3. 嵌入式“灰盒”测试工具

“灰盒”测试是指嵌入式软件既能做“白盒”测试，又能做“黑盒”测试的测试工具，目前主要是基于全数字仿真或半实物仿真技术的应用。

目前在嵌入式测试领域的典型代表是欧洲航天局的 SPACEBEL、SHAM 等产品，国内北京奥吉通科技有限公司的科锐时系列产品 CRESTS/ATAT 和 CRESTS/TESS 等。

4. 嵌入式软件仿真工具

空间飞行器、卫星等工作在太空中，它们的控制软件，即嵌入式软件的调试与测试必须在一个等价太空环境下的仿真环境里进行。

仿真环境的建立需要仿真工具的支持。欧洲航天实时仿真产品 Eurosim 以及网络资源透明访问工具 SPINEware 是最具典型的嵌入式软件仿真工具。

18.2.2 嵌入式软件测试工具举例

1. 美国 Vector 公司的 VectorCAST——单元测试工具

VectorCAST 用于高级语言的单元测试、组装测试及集成测试。它代表了当今软件自动化测试领域的最高水平，它能够：①自动插桩(Stub)及针对被测程序单元自动生成驱动程序；②与主流编译程序器、目标机板，以及实时操作系统(RTOS)相结合；③自动生成小、中、大三种参数取值的测试用例，自动生成测试用例的范围值；④允许顾客采用图像化的点击界面或直观手写(Scripting)界面来设计测试用例；⑤允许完整的自动回归测试；⑥提供图像化说明、分支(Branch)和 MC/DC 代码覆盖率；⑦在主机、仿真器和嵌入式目标机系统上执行测试；⑧支持 Ada 语言和 C/C++等高级语言。

2. IBM Rational Test RealTime(RTRT)

RTRT 是一个针对单元测试和实时分析的交叉平台解决方案，是特别为那些嵌入式、实时软件和其他类型的交叉平台软件产品而开发设计的。

RTRT 使得开发人员在调试过程中能更多地了解代码的执行情况，让开发人员能够在程序运行前修改他们的代码。

使用 RTRT，我们能够：①在开发过程中测试、分析和解决问题；②在主机和目标机上测试和调试；③构造模型驱动测试工具。

3. 美国 FreeScale 公司的 CodeTest——“白盒”测试工具

CodeTest 是专为嵌入式系统软件测试而设计的“白盒”测试工具套件，为追踪嵌入式应用程序、分析软件性能、测试软件的覆盖率以及监控内存动态分配等提供了一个实时在线的高效率解决方案。

CodeTest 还是一个可共享的网络工具，它将给整个开发和测试团队带来高品质的测试手段，并可同时监视整个应用程序，这就避免了在选择程序的哪些部分来观测以及如何配置相应工具来对各部分进行测试时带来的困难。即便是在程序超出高速缓存(Cache)或内存被动态再分配时，CodeTest 仍能生成可靠的追踪及测试结果。

在进入连续运行模式时，CodeTest 能够同时测试出软件的性能、代码覆盖以及存储器动

态分配，捕获函数的每一次运行。

无论是在检测一个局部的软件模块还是整个软件系统测试，工程师只需简单地将CodeTest的仿真探头（probe）插到目标系统的处理器上，预处理待测的源程序，启动CodeTest，运行测试处理软件。

测试结果在测试进行过程中或在测试结束后均可随时观看。

4. 比利时SPACEBEL公司产品——全数字仿真测试工具

比利时SPACEBEL公司的ERC32/1750/ADSP Target Simulator用于航天设计、仿真应用等方面。目标机模拟器能够模拟ERC32/1750/ADSP计算机，如能够模拟CPU指令、各种硬件动作和I/O。

它所提供的功能比仿真器更加丰富，如：能够进行中断和错误注入；当程序执行到断点处时，仿真硬件全部"冻结"，从而便于实时调试；支持如IU、FPU、UART、timers、DMA、ATAC以及各类I/O活动；时间性能比较理想。

在进行"白盒"测试时能够给出被测试代码执行的覆盖情况，并且完全不需要实施插桩。

ERC32/1750/ADSP Target Emulator能够与配备了Jtag接口的ERC32目标板相连，能够访问所有的IU、FPU和MEC寄存器，以及目标板上所有的存储单元。

5. 荷兰CHESS公司产品SHAM——半实物仿真测试工具

荷兰CHESS公司星载软件验证设备SHAM是专门针对航天工业中卫星上的姿轨控制系统和数据管理系统的嵌入式软件验证（确认）开发的专用系统。

该系统是一个多计算机环境，由宿主机系统和仿真处理模块SHAM组成。SHAM包含一个目标处理器和支持与控制系统。目标处理器执行被测汇编语言程序，Ada语言程序，汇编语言、Ada语言混合语言程序的最终二进制代码；而支持与控制系统则控制目标处理器的行为并仿真低级硬件接口。宿主机用于应用测试和全面控制，并提供更复杂环境下的仿真模拟。无须对被测汇编语言程序、Ada语言程序、混合语言程序最终二进制代码进行任何修改，可直接在真实目标处理器中执行，应用于被测软件真实的外界感知环境的仿真系统中。SHAM支持运行在真实目标处理器基础上的覆盖率分析。

SHAM已经几次检测出严重的星载软件的缺陷，及时发现了被测星载软件的失效过程，防止了整个卫星的损失。

6. 北航的"黑盒"测试工具GESTE

GESTE（General Embedded System Testing Environment）为嵌入式系统提供仿真测试环境，实现对嵌入式系统进行实时、闭环、非侵入式的系统测试。

该产品的主要功能有：测试环境与测试仿真模型开发、测试脚本图形化生成/调试、测试环境配置、实时测试、测试数据收集定制、测试数据事后分析及日志管理等。

GESTE采用当前主流嵌入式操作系统VxWorks，定时精度为1ms，根据仿真环境复杂程度不同，可以满足工作周期不小于5ms的实时嵌入式系统测试。

针对用户的不同需求，该产品具有较强的适应能力，可针对不同的用户进行定制，提供开放的测试脚本描述语言及仿真建模环境，充分满足用户在系统测试、开发中的各种需求。

7. 欧洲航天局实时仿真产品Eurosim

Eurosim是一个可构造的模拟仿真系统工具，应用于太空项目和工业项目生存周期所有阶段的人机交互和设备交互实时仿真。应用Eurosim可以使已有的模型软件重用。

每个模拟过程都可以分解成多个可重用的部件工具，然后对每个部件工具的功能细节进行仿真。详细设计的工具部件可重复应用到小规模以及规模庞大的仿真应用中。

Eurosim 可以帮助用户减少模拟仿真所花费的费用，使在工程项目中可以尽早和更广泛地进行模拟仿真活动（如可行性评估阶段、设计阶段、原型构造阶段、生产/测试阶段、操作/培训阶段）。

Eurosim 提供一个开放的系统结构，与 Matlab、PvWave、Vega、Satellite Tool Kit（STK）集成使用，可以更好地发挥 Eurosim 及相关工具的功效。

相关应用涉及从子系统到整个系统的系统设计、系统分析、可视化设计、系统仿真设计。

Eurosim 已经应用于多个太空项目、工业项目。如：全球导航卫星的性能验证、自动导航水下交通工具、Euromoon 2000-月球登录车、臭氧监控器等的仿真模拟、TVE-软件验证设备、战斗机飞行员模拟仿真训练系统。

8. 网络资源透明访问工具 SPINEWare

当前每个用户都面临着与越来越复杂、越来越多样化的计算机系统打交道，并且要求这些计算机系统在一个分布式的网络环境下工作。

在空间科学的研究中，在系统仿真阶段面临着和各种仿真工具打交道，处理、应用和转换网络中各种各样的数据，形成工作产品的难度越来越大。

规范仿真工作中的工作流程，为工作流程建模成为一种自然而然的需要。

SPINEWare 是一个中间件系统，它通过提供一组集成工具和中间件支持用户在由不同计算机平台所组成的计算机网络中构建工作环境，并在工作环境中运用一致的、完整的、可视化的操作使用有关资源（对象资源）。

SPINEWare 的工作环境为用户提供了访问网络资源的能力，这些资源就像存放在一台虚拟的计算机中，在该计算机中，计算、存储、输入/输出以及信息（如软件、数据与文档）等资源都是通过树结构以对象的形式提供给用户进行访问的，用户通过图形用户界面能方便地操作这些对象。

当前的仿真工作可能需要大量的不同性质的资源，仿真设计人员可能并不是某个领域或者某个软件的使用专家，但构建特定的仿真过程需要这些资源。SPINEWare 的工作流程建模器，提供简单方式定义工作流程，应用工作流程的能力，使特定的业务流程规范化、模型化。

18.2.3 传统测试工具的局限性

传统测试工具在这里主要指“白盒”测试工具和“黑盒”测试工具。

1. 传统“白盒”测试工具缺陷

传统“白盒”测试工具具有如下缺陷。

（1）所有的结构测试都要求插桩，造成被测软件代码膨胀。代码膨胀使本来就匮乏的系统资源更加紧张。尤其是汇编程序，由于它是低级语言，即它的结构化能力弱，指令功能低级，所以要构造复杂的算法，需要大量的汇编子程序调用、条件判断以及程序跳转指令，插桩后代码膨胀更为明显。

代码膨胀有可能导致：①系统错误（被测程序设计中的代码和数据分配受到影响）、时序错误（被测程序的中断与端口输入/输出的时序延时），甚至逻辑错误（汇编程序中相对调用或跳转的目的地址可能发生变化）；②影响软件运行的真实性和实时性，无法对软件运行起来后进行实时跟踪。

（2）外部事件的激励很难引入。中断事件、输入/输出事件以及其他相关事件无法按逻辑时序产生，无法构造能使被测软件闭环运行的测试环境。

（3）占用硬件资源。基于宿主机/目标机的工作方式的“白盒”测试工具其目标机的地址

空间难以做到对用户全部开发(部分被占用)。

2. 传统"黑盒"测试工具缺陷

传统"黑盒"测试工具具有如下缺陷:

(1) 被测软件的运行环境目标硬件必须存在,这在目标硬件还没有开发出来,或目标硬件发生变化,甚至被测方不提供目标硬件的情况下是无法测试的。

(2) 价格非常昂贵。

(3) 外部逻辑信号的产生逻辑需要程序员编程,需要计算机处理(专用计算机)。

(4) 多路信号的配置可能满足不了我们实际应用变化的要求。

(5) 维护困难,易损坏。

18.3 全数字仿真测试方案

18.3.1 全数字仿真的概念

嵌入式软件的全数字仿真就是脱离目标机,用数字模拟硬件或电路的信号结果并交给模拟软件计算和处理,如图 18-2 所示。其中模拟或解释执行 CPU 指令和相关时序从而避免了插桩。

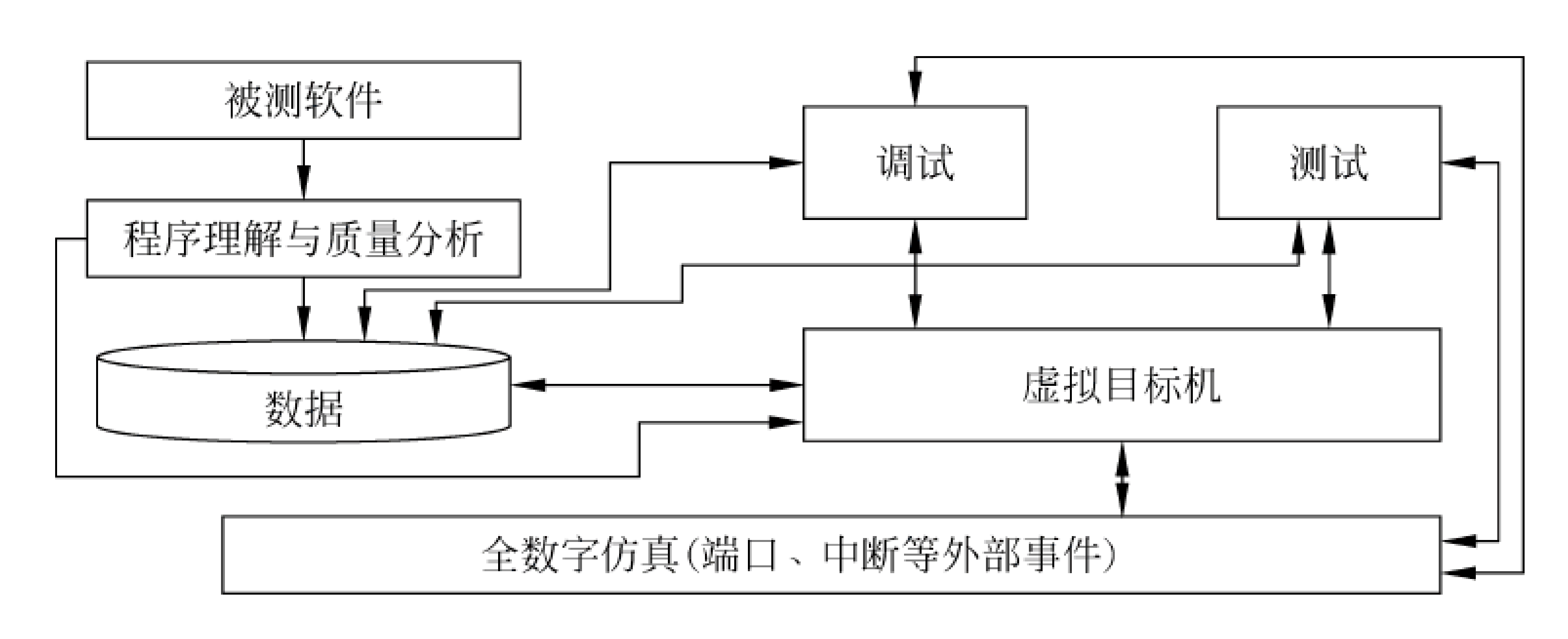


图 18-2 嵌入式软件全数字仿真测试框图

SPACEBEL 公司的 ERC32 Target Simulator 和北京奥吉通的 CRESTS/ATAT 与 CRESTS/TESS 就是嵌入式软件全数字仿真的典型代表。

18.3.2 北京奥吉通的 CRESTS/ATAT 和 CRESTS/TESS 介绍

北京奥吉通的全数字仿真产品 CRESTS/ATAT 和 CRESTS/TESS 通过虚拟目标机解释执行嵌入式软件和对外围电路及外部事件进行全数字化仿真,很好地解决了前面我们提到的代码膨胀问题和硬件环境无法搭建问题。

尽管嵌入式软件在虚拟目标机环境的运行效率较低,但现在计算机配置越来越高,性能越来越好,速度越来越快,内存越来越大,运行效率已经不再是主要问题了。

CRESTS/ATAT 和 CRESTS/TESS 是分别针对汇编语言和高级语言的分析与测试工具,为嵌入式系统提供全数字仿真测试环境或测试平台,实现对嵌入式系统进行实时、闭环的、非侵入式(不插桩)的系统测试。

在该平台上能够对被测软件进行静态分析、模拟运行、高级调试和综合测试,实现了嵌入式软件外部事件的全数字仿真,嵌入式软件就像在真实硬件环境下连续不中断地运行。

CRESTS/ATAT 和 CRESTS/TESS 的工作流程是:首先装载在开发环境中交叉编译后

的被测软件；然后对被测程序进行静态分析，生成程序理解和质量度量的数据；接着，对被测嵌入式软件程序进行测试和调试，此时，可通过全数字仿真模拟端口、中断等外部事件，使被测嵌入式软件程序能够"闭环"运行，实现测试的自动化；最后，对测试结果进行分析，生成测试报告。

依据上述工作流程，我们可以清楚地看到 CRESTS/ATAT 和 CRESTS/TESS 能够满足嵌入式软件开发阶段的内部测试和调试，以及验收阶段的验收测试的要求，能够为测试方、被测方及上级主管单位提供可以信赖和可以再现测试过程与测试问题的测试报告。

1. CRESTS/ATAT

该工具目前支持 Intel 8031/8051/8052、8096/80196、80x86，Mil-1750 及 DSP TMS320C2X/C3X/C4X/C5X 等 CPU 的汇编程序测试，并为汇编语言的测试提供了有效、统一的协同工作平台。在该平台下能够完成汇编程序的分析与检查、汇编代码的运行与调试、汇编单元的配置与测试、汇编系统的仿真与测试、中文测试报告生成。有效结合了测试与调试的能力，规范了汇编语言的测试流程。

CRESTS/ATAT 汇编测试工具分析与检查功能支持代码编程规则检查，并对影响程序结构化的代码进行警告；提供程序控制流图、DD 路径图、程序调用树、程序被调用树和程序危害性递归等；给出度量程序质量的多种度量元（如 McCabe 的圈复杂度、程序跳转数、程序扇入扇出数、程序注释率、程序调用深度、程序长度、程序体积、程序调用及被调用描述等）。

CRESTS/ATAT 的汇编代码运行与调试功能为汇编用户提供了不需真实硬件的 CPU 模拟运行环境。在该环境下，解释执行所有的 CPU 指令，模拟其指令时序，模拟定时中断等；支持汇编程序的各种调试，包括控制程序运行方式、修改程序运行状态、观察程序运行结果等。

CRESTS/ATAT 的 CPU 上下文场景的自编程配置能力解决了对汇编程序进行单元测试的需求，用户可根据单元测试的要求，灵活方便地对 CPU 上下文场景进行配置，形成汇编程序单元执行的驱动。

汇编系统仿真与测试功能提供了对汇编程序进行功能测试与覆盖测试的手段。其中覆盖测试支持汇编程序的语句、分支和调用覆盖测试，并支持图形化显示；而外部事件的编程仿真方式，解决了外部激励、系统闭环运行和功能测试的要求。

中文测试报告以超文本形式给出了被测汇编程序的静态分析和动态测试的各种结果及结果统计，提出了汇编编程风格约定和汇编程序质量度量建议。

CRESTS/ATAT 的外部事件仿真是通过用 TCL 高级脚本编程模拟 I/O 与中断事件的产生。被测汇编程序在模拟环境运行过程中，尽管存在大量的 I/O 与中断事件产生的要求，也能够与真实硬件环境一样连续不中断地运行。这在设计初期，真正的硬件没有制造出来，或设计发生变化，硬件来不及改变，以及对软件进行维护等，全数字仿真测试的方法是非常重要的。

端口 I/O 与中断事件产生的自编程模拟功能很好地解决了汇编程序在模拟运行环境下的闭环测试问题，实现了测试过程的自动化。

被测汇编程序的测试用例可用 TCL 脚本语言编写和管理。

从上我们看出，CRESTS/ATAT 支持各种测试需求。端口 I/O 与中断事件产生的自编程模拟功能以及 CPU 上下文场景的自编程配置能力为用户提供了"黑盒"测试及单元测试的手段。全数字指令模拟执行技术支持不用插桩的"白盒"测试。CRESTS/ATAT 支持汇编程序检查分析、汇编代码运行调试、"白盒"测试、"黑盒"测试以及单元与集成测试等，支持"灰盒"测试技术的应用。

2. CRESTS/TESS

由于CRESTS/TESS是针对高级语言C的全数字仿真测试，其技术含量最大的是程序分析技术和符号解释技术。CRESTS/TESS所虚拟的目标机是TI的DSP TMS320 C3X/C4X，即通过软件模拟的方式实现TI公司的DSP TMS320 C3X/C4X的虚拟目标机。虚拟目标机要完成的任务有CPU指令集的解释、CPU时序的模拟、CPU端口动作的仿真、CPU中断机制，以及CPU流水、缓冲和并行指令等。

CRESTS/TESS在程序理解方面要做的工作是解决C语言函数之间的调用关系、被调用关系以及函数内部的控制流程关系的表示和图形显示。

CRESTS/TESS软件质量度量方面是以国际软件质量标准ISO/IEC 9126和权威理论为基础，给出那些严重影响程序整体质量的度量元，实现McCabe的圈复杂性度量，以及Halstead源代码复杂性度量等。

CRESTS/TESS在软件测试方面支持软件测试常用的覆盖测试、功能测试、单元测试、系统测试及回归测试；支持测试用例或测试脚本的应用，支持“闭环”测试和测试过程的自动化；对于覆盖测试支持调用覆盖、分支覆盖以及语句覆盖的图形显示。

CRESTS/TESS支持高级符号调试功能，能够有效地控制程序运行，观察或改变程序运行状态。实现了程序的单步运行、连续运行、设置断点等手段控制程序的运行，实现了代码、数据、寄存器内容以及高级符号变量的读写或改变程序的运行状态。

CRESTS/TESS提供模拟外部设备产生外部激励信号的机制（全数字仿真），即用TCL脚本语言编写端口事件、中断事件以及其他外部事件的逻辑流程。

CRESTS/TESS能够自动生成软件分析与测试总结报告。所生成的软件分析与测试总结报告是超文本的，它给出被测程序的程序理解信息、质量度量信息、程序运行信息以及测试结果统计信息等。

习题

1. 嵌入式软件有哪些特点，嵌入式软件测试与传统软件测试相比有哪些不同之处？
2. 简述嵌入式软件测试的传统方法和嵌入式软件测试的流程。
3. 说明嵌入式软件传统测试方法的局限性，并简述其原因。
4. 以CRESTS/ATAT为例简述全数字仿真的思想和全数字仿真测试的概念。